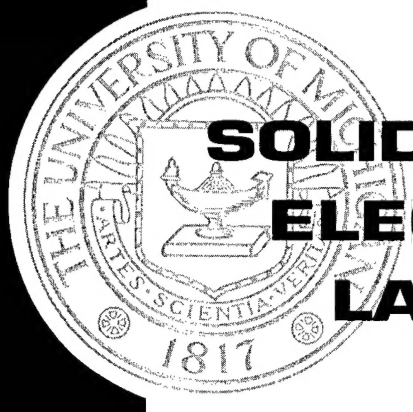
A MICROARCHITECTURE FOR RESOURCE-LIMITED SUPERSCALAR MICROPROCESSORS

TECHNICAL REPORT NO. SSEL-289

1999

By

Todd David Basso



SOLID-STATE ELECTRONICS LABORATORY

DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE
THE UNIVERSITY OF MICHIGAN, ANN ARBOR

This report has also been submitted as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the University of Michigan, 1999.

SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1999	Technical

4. TITLE AND SUBTITLE
A Microarchitecture for Resource-Limited Superscalar Microprocessors

5. FUNDING NUMBERS
DAAH04-94-G-0327

6. AUTHOR(S)
Todd David Basso

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
University of Michigan
Department of Electrical Engineering
1301 Beal Ave.
Ann Arbor, MI 48109-2122

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 33790.75-EL

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Microelectronic components in space and satellite systems must be resistant to total dose radiation, single-event upset, and latchup in order to accomplish their missions. The demand for inexpensive, high-volume, radiation hardened (rad-hard) integrated circuits (ICs) is expected to increase dramatically as the communication market continues to expand. Motorola's Complementary Gallium Arsenide (CGaAs^{TM}) technology offers superior radiation tolerance compared to traditional CMOS processes, while being more economical than dedicated rad-hard CMOS processes.

The goals of this dissertation are to optimize a superscalar microarchitecture suitable for CGaAs^{TM} microprocessors, develop circuit techniques for such applications, and evaluate the potential of CGaAs^{TM} for the development of digital VLSI circuits.

14. SUBJECT TERMS	15. NUMBER IF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ABSTRACT

A MICROARCHITECTURE FOR RESOURCE-LIMITED SUPERSCALAR MICROPROCESSORS

by

Todd David Basso

Chair: Richard Brown

Microelectronic components in space and satellite systems must be resistant to total dose radiation, single-even upset, and latchup in order to accomplish their missions. The demand for inexpensive, high-volume, radiation hardened (rad-hard) integrated circuits (ICs) is expected to increase dramatically as the communication market continues to expand. Motorola's Complementary Gallium Arsenide (CGaAs™) technology offers superior radiation tolerance compared to traditional CMOS processes, while being more economical than dedicated rad-hard CMOS processes.

The goals of this dissertation are to optimize a superscalar microarchitecture suitable for CGaAs™ microprocessors, develop circuit techniques for such applications, and evaluate the potential of CGaAs™ for the development of digital VLSI circuits.

Motorola's 0.5 μm CGaAs™ process is summarized and circuit techniques applicable to digital CGaAs™ are developed. Direct coupled FET, complementary, and domino logic circuits are compared based on speed, power, area, and noise margins. These circuit techniques are employed in the design of a 600 MHz PowerPC™ arithmetic logic unit. The

dissertation emphasizes CGaAs™-specific design considerations, specifically, low integration level.

A baseline superscalar microarchitecture is defined and SPEC95 integer benchmark simulations are used to evaluate the applicability of advanced architectural features to microprocessors having low integration levels. The performance simulations center around the optimization of a simple superscalar core, small-scale branch prediction, instruction prefetching, and an off-chip primary data cache. The simulation results are used to develop a superscalar microarchitecture capable of outperforming a comparable sequential pipeline, while using only 500,000 transistors. The architecture, running at 200 MHz, is capable of achieving an estimated 153 MIPS, translating to a 27% performance increase over a comparable traditional pipelined microprocessor.

The proposed microarchitecture is process independent and can be applied to low-cost, or transistor-limited applications. The proposed microarchitecture is implemented in the design of a 0.35 μm CMOS microprocessor, and the design of a 0.5 μm CGaAs™ microprocessor. The two technologies and designs are compared to ascertain the state of CGaAs™ for digital VLSI applications.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER 1

INTRODUCTION

Our society is becoming highly dependent upon satellites. We rely upon satellites for everything from cellular phone conversations to military defense systems. The failure of the Telstar 401 satellite in January of 1997 underscores how vulnerable microelectronics in space are to the effects of solar disturbances. The number of communication satellite licenses granted by the FCC indicate that over one thousand satellites could be launched into low-earth-orbit over the next five years [1]. Microelectronic components in these systems must be resistant to total dose radiation, single-even upset, and latchup in order to accomplish their missions.

Radiation hardened (rad-hard) electronics have been operating in space since the beginning of the space program decades ago, but the demand for inexpensive, high-volume rad-hard ICs is expected to increase dramatically as the communication market continues to expand. Traditional CMOS processing can be enhanced to render the IC radiation tolerant, but the cost of rad-hard CMOS ICs can be 10 to 100 times that of commercial CMOS ICs. In light of the increased cost, alternative technologies such gallium arsenide must be considered. Motorola's Complementary Gallium Arsenide (CGaAs™[1]) technology operates at low voltages, providing a power savings over CMOS, while also being

1. CGaAs is a trademark of Motorola.

innately rad-hard. CGaAs is also a relatively low-cost alternative; the cost of a CGaAs IC is approximately 5 times that of a similar high-volume commercial CMOS IC [2], making it significantly less expensive than that of a rad-hard bulk CMOS IC.

This thesis describes the development of a CGaAs PowerPC™[1] microprocessor suitable for space applications. Motorola's 0.5 μm CGaAs process was selected as the semiconductor technology because of its unique abilities to meet the requirements of space and satellite systems. Recently published findings [2] indicate that CGaAs has a power-delay product of 0.01 μW/MHz/gate. Additionally, the CGaAs process is resistant to single-event upset (SEU) (10^{-10} upsets/bit-day), total dose radiation (10^8 rads), and latchup (10^{12} rads). These properties make CGaAs attractive for space and satellite applications. However, CGaAs presents design challenges such as reduced power-supply voltage, gate leakage, subthreshold drain-source leakage, and a low transistor integration level, all of which impact digital circuits. The processor architecture described in this thesis is driven by the constrained integration level. Architectural simulations reveal that an acceptable level of performance can be reached by efficiently utilizing the resources available. The processor implements a small on-chip primary instruction cache backed by a two-entry stream buffer and a larger off-chip primary data cache. The instruction fetch mechanism is guided by a small two-level dynamic branch prediction mechanism. Computation is performed by a small superscalar execution core. The architecture, running at 200 MHz, is capable of achieving an estimated 153 MIPS, translating to a 27% performance increase over a comparable traditional pipelined microprocessor.

---

1. PowerPC Architecture, POWER Architecture, PowerPC 601, PowerPC 603, PowerPC 604, PowerPC 620, and PowerPC 750 are trademarks of IBM.

1.1 Thesis Overview

This thesis will summarize the design challenges of developing a radiation-hard CGaAs PowerPC microprocessor. The constraints and limitations of the technology as they affect system architecture will be discussed. Architectural features that improve the performance of the processor despite the limited integration levels will be presented. Implementation details for two microprocessors, derived from the proposed architecture, will be given.

Chapter 2 describes various aspects of gallium arsenide (GaAs) technology. Previous GaAs microprocessor development at The University of Michigan, centered around GaAs E/D MESFET technology, is summarized. An overview of Motorola's 0.5 μm CGaAs process is given. The design challenges presented by the CGaAs technology are discussed. Several circuit design techniques applicable for developing digital CGaAs ICs are introduced and compared. A prototype arithmetic logic unit designed with CGaAs domino logic is described.

The PowerPC architecture provides the architectural framework for this research. Chapter 3 describes the philosophy of the PowerPC architecture itself, and our motivation for adopting it as our platform. The beginnings of IBM's POWER architecture through the evolution of the current PowerPC Instruction Set Architecture (ISA) are summarized. Implementations of several PowerPC microprocessors are described. These implementations demonstrate the flexibility of the PowerPC ISA. The architectural philosophy of the CGaAs microprocessor is derived from the PowerPC architecture.

Researchers at the University of Michigan are engaged in the design of a PowerPC microprocessor system, based upon CGaAs and multichip module (MCM) technologies, referred to in the literature as the PowerPC Ultrafast Multichip Module Architecture (PUMA) project. The goal of the project is to investigate the technological design challenges and performance potential of the combination of CGaAs and MCM technologies. Chapter 4 gives a brief overview of the PUMA project. The role of the MCM in the system architecture is discussed, placing the design of a CGaAs microprocessor in the proper context.

The primary design challenge presented by the CGaAs process is the low transistor integration level. Chapter 5 is a performance study dealing with various aspects of microarchitecture. Performance enhancing features are proposed and the resulting performance is analyzed. High-performance architectural concepts are applied on a small scale and simulated using a PowerPC cycle-level simulator. The goal is to construct an architecture that can meet the limited transistor budget, while still offering reasonable performance. The chapter concludes with a performance comparison of several microprocessor configurations having various transistor budgets.

The findings of Chapter 5 are used to propose a microarchitecture for the CGaAs PowerPC integer processor, or fixed-point unit (FXU), central to the development of the PUMA system. Chapter 6 formally proposes a microarchitecture for the CGaAs PowerPC microprocessor. A detailed description of the FXU pipeline is presented. The functions of key pipeline stages and essential hardware components are described.

Chapter 7 describes the implementation details associated with the physical design of the microprocessor. The microarchitecture was first implemented in a traditional CMOS process, and then in the targeted CGaAs process. Both implementations are described and characterized. Comparisons are drawn between CMOS and CGaAs. The present state of the CGaAs technology for the design of digital VLSI circuits is evaluated. Recommendations are made to improve the CGaAs technology.

The final chapter, Chapter 8, presents my conclusions and major research contributions. Directions for future research are proposed.

CHAPTER 2

COMPLEMENTARY GALLIUM ARSENIDE TECHNOLOGY

This chapter discusses the use of Motorola's Complementary Gallium Arsenide (CGaAs) technology for implementing digital VLSI circuits. CGaAs was selected as our semiconductor technology because of its high-speed, good power-delay product, and radiation-hard properties. Historically, the Aurora project at The University of Michigan has focused upon MIPS processors built with Gallium Arsenide Enhancement/Depletion Metal Semiconductor Field Effect Transistor (GaAs E/D MESFET) technology. In this chapter we summarize the previous research performed by the Aurora project and present CGaAs, the current semiconductor technology.

2.1 History of GaAs at The University of Michigan

The Aurora project at The University of Michigan centered around the development of MIPS processors using the Vitesse GaAs E/D MESFET technology. The project spanned five years and was divided into three distinct phases. The first phase was a feasibility study which proved that a GaAs processor could indeed be built. The second phase focused upon increasing the clock frequency. The final phase was an attempt to implement an architecturally aggressive microprocessor. Table 2.1 summarizes the three processors developed during the project and compares each processor with a typical CMOS micro-

Microprocessor (Year)	GaAs		ISSCC (CMOS)	
	Clock frequency (MHz)	Transistor Count (M)	Clock frequency (MHz)	Transistor Count (M)
Aurora I (1991)	125	0.060	83	0.920
Aurora II (1993)	190	0.160	110	2.000
Aurora III (1995)	300	0.500	150	4.400

Table 2.1: GaAs RISC processors developed at The University of Michigan.

processor (represented by the average frequency and transistor count) from ISSCC in each corresponding year. The table shows that the GaAs processors were nearly 40% faster than their CMOS counterparts, but integrated only 10% as many transistors.

2.1.1 Aurora I

The Aurora I processor [3] was the first of three GaAs processors designed at the University of Michigan. The processor was fabricated in the Vitesse HGaAs II technology (1.2 μm with three metal layers) and implemented a 28-instruction subset of the MIPS R2000 architecture. The chip was designed as a demonstration vehicle, and did not contain on-chip caches nor support exceptions. The limited functionality also guaranteed that the pipeline would not be required to stall, thus the control logic was simple. The Aurora I microprocessor integrated 60,000 transistors, reached an operating frequency of 137 MHz, and dissipated 11 W with a 2 V supply. These results demonstrated that a GaAs microprocessor was feasible.

2.1.2 Aurora II

The primary goal of Aurora II [4] was to explore issues in high-frequency microprocessor design. The processor was fabricated in the Vitesse HGaAs III DCFL technology (1.0 μm with four metal layers). The Aurora II processor implemented additional func-

tionality over the Aurora I design, including shift instructions, a cache interface, and exception support. The chip implemented a two-phase clocking scheme. The Aurora II design consisted of 160,000 transistors. Aurora II operated with a 2 V supply and consumed 24 W. Portions of the chip were found to operate at 200 MHz. Design errors limited other portions of the processor to only 100 MHz. The limited functionality of Aurora II made it difficult to draw conclusions about clock frequency and the performance potential of the GaAs E/D MESFET technology.

2.1.3 Aurora III

The Aurora III integer processor unit (IPU) [5] was the culmination of the GaAs E/D MESFET development effort, and represented a significant increase in complexity over Aurora I and Aurora II. The Aurora III IPU was an aggressive dual-issue superscalar implementation of the MIPS R3000 instruction set architecture, as opposed to the classic 5-stage MIPS pipeline implemented in the earlier Aurora processors. The pipeline was similar to the Motorola 88110 [6]. The IPU was designed to interface with an external pipelined primary data cache, floating-point unit, and memory management unit. The designs made use of queues at the inputs, implementing a decoupled architecture. To achieve a high clock rate, the Aurora III used a simple-issue model similar to the Alpha 21064 [7]. The processor consisted of an on-chip bus interface unit, dual integer execution units, instruction fetch unit, load-store unit, and a prefetch unit. Performance simulations estimated that Aurora III would achieve a CPI of 1.25. Electrical simulations predicted that the 500,000 transistor Aurora III microprocessor would operate at 300 MHz and consume 48 W with a 2 V supply.

2.1.4 GaAs E/D MESFET Technical Challenges

The GaAs E/D MESFET technology presented several design challenges to the Aurora designers. The high source resistance of GaAs MESFETs limited the use of series enhancement transistors, which are required to build NAND-based structures. The device sizes required to enable NAND-based logic gates to meet the required noise margins rendered the gates too slow. The inability to place transistors in series forced the designers to use NOR-only logic structures. The end result was that more levels of logic were required to implement a given function. A typical logic function required 15% more levels of logic in GaAs DCFL than CMOS [5], reducing the switching speed advantage held by GaAs devices. Designers of the Aurora processors found that in general, a GaAs implementation required 91% more gates than an comparable CMOS implementation [5].

The metal-semiconductor interface of a GaAs E/D MESFET transistor forms a Schottky barrier diode between the channel and gate. The diode clamps the logic high voltage to 0.7 V, causing the gates to have a small logic swing. While this small logic swing is responsible for some of the speed advantage of GaAs circuits, it is a disadvantage in that it reduces noise margins. In the Aurora project, the output low voltage (V_{OL}) was chosen to be one-half of the enhancement device threshold. To ensure that the output would reach V_{OL}, the pulldown transistor was required to be 3 to 4 times stronger than the pullup device, resulting in large transistor widths. In the HGaAs III process, the enhancement mode transistors had a threshold voltage of 0.22 V, nearly 1/3 of the logic swing. This proportionately large threshold voltage further degraded the gate switching speed.

2.2 Complementary GaAs Technology

Now, having examined the previous research done by the Aurora group, we turn our attention to the CGaAs technology.

2.2.1 Device Characteristics

The application of CGaAs, a complementary heterostructure-insulated-gate field effect transistor (HIGFET or HFET) technology for high-speed VLSI circuits has been described recently [2]. A sketch of the device structure is shown in Figure 2.1. CGaAs integrates an enhancement-mode p-type transistor with a high performance n-type transistor. Historically, the primary interest in GaAs and other III-V materials has been their high electron mobilities. While holes in III-V materials do not enjoy an intrinsic mobility advantage over those in silicon, the pseudomorphic p-type transistors in this process have three to five times higher transconductances at given gate dimensions than their silicon counterparts. The current CGaAs process has three levels of Al-Cu interconnect. Standard gate

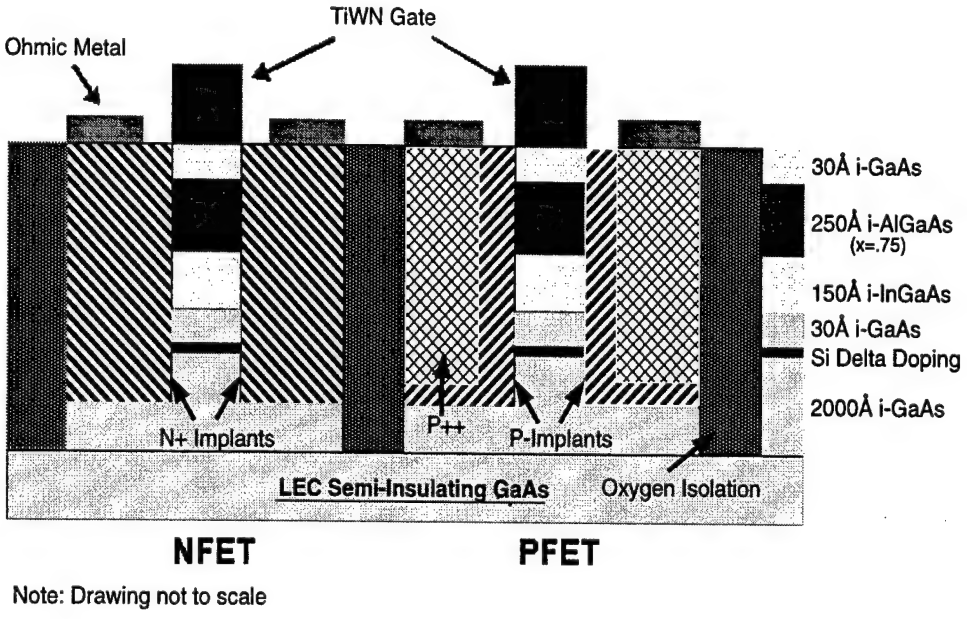


Figure 2.1: CGaAs process cross-section [2].

lengths are now 0.7 μm and 0.5 μm; typical parameters for 0.7 μm channel-length devices with ±0.55 V thresholds (measured with Vdd = 1.5 V) are given in Table 2.2. As seen in the table, both n-type and p-type transistors have good output conductances and pinch-off characteristics.

Parameter	n-type HFET (0.7 x 10 μm)	p-type HFET (0.7 x 10 μm)
V_T (V)	+0.55	-0.55
$I_{D,sat}$ (mA)	1.8	0.5
g_m (mS/mm)	280	60
Beta (mA/V^2-mm)	270	50
Subthreshold slope (mV/dec)	75	90
Subthreshold Current (nA) (V_{GS}=0V)	< 1	< 10

Table 2.2: CGaAs device parameters [2].

2.2.2 Radiation Tolerance

A Low Temperature GaAs (LTG) layer is used in the HFET structure to guard against charge collection from ionizing particle radiation [8]. Its epitaxial structure makes the CGaAs process resistant to single-event upset (SEU) and latchup. Complementary logic in CGaAs is resistant to SEU with fewer than 10^{-10} upsets/bit-day and it will not exhibit latchup, even beyond 10^{12} rads. The absence of a gate oxide and field oxide in CGaAs makes it resistant to total dose radiation up to 10^{8} rads.

2.2.3 Power-Delay Performance

Fig. 2.2 shows unloaded ring oscillator gate delays versus supply voltage for several logic families with ±0.55 V device thresholds [2]. The delay of 1.0 μm CGaAs is less than that of commercial 0.5 μm CMOS or thin-film silicon-on-insulator (TFSOI), and 0.5 μm CGaAs shows delays below 100 ps with a 1.2 V power supply. Circuits used for process

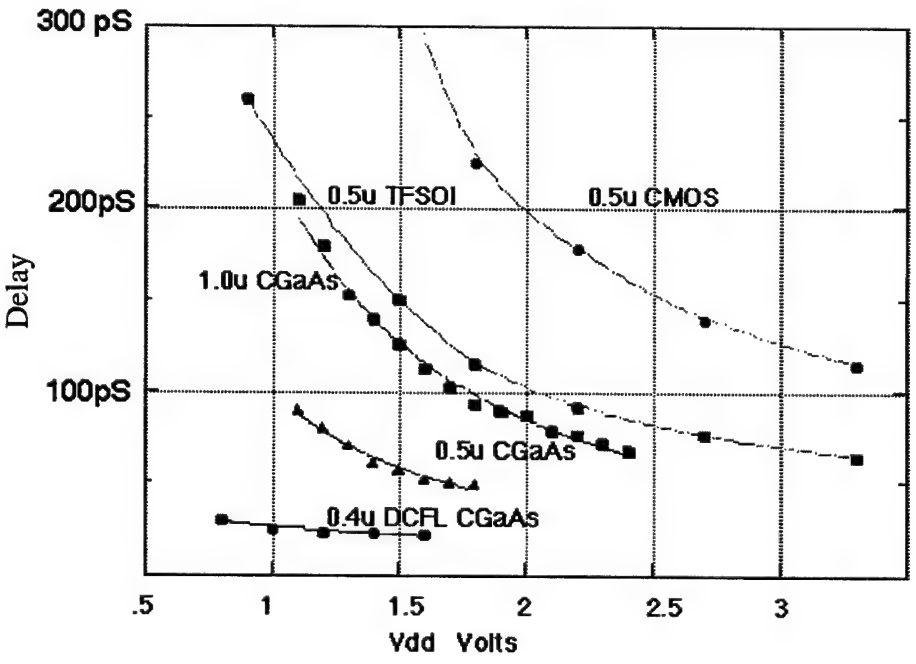


Figure 2.2: Propagation delay versus supply voltage for various technologies [2].

monitoring, such as a 32-bit shift register, have demonstrated a power-delay performance of 0.01 μW/MHz/gate at 0.9 V and 0.1 μW/MHz/gate at 1.2 V.

2.2.4 CGaAs Technical Challenges

Two key parameters of concern in the CGaAs process are gate leakage and subthreshold drain-source leakage, which determine the stand-by power dissipation of complementary circuits. Gate leakage also affects power, area, and speed. Gate leakage causes IR drops along the power rails, requiring wider rails to decrease the resistance and thus reduce the IR drop. Wider power rails "bloat" the design, increasing delay. Drain-induced barrier lowering (DIBL) increases gate current when the drain-to-source voltage is high, as is the case when a logic input changes state.

A further design challenge results from the low power supply voltages. Low supply voltages reduce power consumption. However, when coupled with comparatively high device thresholds (±0.55V) the drain currents of both n-type and p-type transistors are reduced. The end result is that more time is required to charge and discharge circuit nodes. This is especially detrimental for p-type transistors, which, because of the lower mobility of holes than electrons in the InGaAs channel, have transconductances only 1/4 those of comparable n-type transistors. This transconductance ratio requires p-type transistors to have wider channels, significantly increasing the input capacitance of complementary logic gates. The high input capacitance and small (V_{GS} - V_T) slow CGaAs circuits. Threshold voltages could be reduced, and in fact, have been reduced on experimental wafers, significantly improving transconductances at the expense of higher leakage currents.

The greatest challenge in designing CGaAs integrated circuits is the low level of transistor integration. Commercial CGaAs circuits of 170,000 transistors have been demonstrated [2], and the integration level has been raised to 400,000 transistors for current designs. Complex ICs, such as microprocessors, require far more transistors. One of the objectives of this research is to develop an architecture for the microprocessor system that adheres to the integration level while maximizing performance.

2.2.5 CGaAs Digital Logic

CGaAs offers several advantages over GaAs E/D MESFET technology. The drain-source resistance of CGaAs devices is much lower than that of GaAs E/D MESFET enhancement transistors, permitting series combinations of transistors. This enables the use of NAND- as well as NOR-style logic circuits. The presence of both n-type and p-type transistors in the CGaAs technology enables a variety of logic families to be implemented. The various logic families afford designers the ability to make trade-offs with respect to speed, power, area, and supply voltage for specific applications. Here we summarize several common digital circuit techniques used in the PUMA research project.

2.2.5.1 Direct-Coupled FET Logic (DCFL)

Like GaAs E/D MESFET design, it is possible to use Direct-Coupled FET Logic (DCFL) to attain high switching speeds. A DCFL circuit implementing a two-input NAND logic function is shown in Figure 2.3. The distinguishing feature of DCFL circuits is the active load. The gate of the p-type transistor is grounded, ensuring that the device is always active. The presence of the active load results in a number of inefficiencies. DCFL

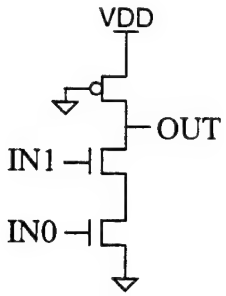


Figure 2.3: DCFL NAND2 circuit implementation.

gates dissipate a great deal of power, since the load is conducting while the series n-type transistors are attempting to pull the output node low. This creates a conducting path between VDD and GND in the output low state. DCFL circuits are ratioed logic, meaning that the transistors must be sized properly for the circuit to function correctly. The resulting V_{OL} is relatively high because of the active load constantly pulling toward V_{DD}.

Figure 2.4 shows the HSPICE [9] simulation for the DCFL NAND2 circuit. In this simulation, the n-type transistor width was varied such that the ratio of n-type to p-type transistor widths ranged between 1 and 6. The simulation demonstrates that DCFL is a ratioed logic family. When the ratio of channel widths is one, the circuit is unable to

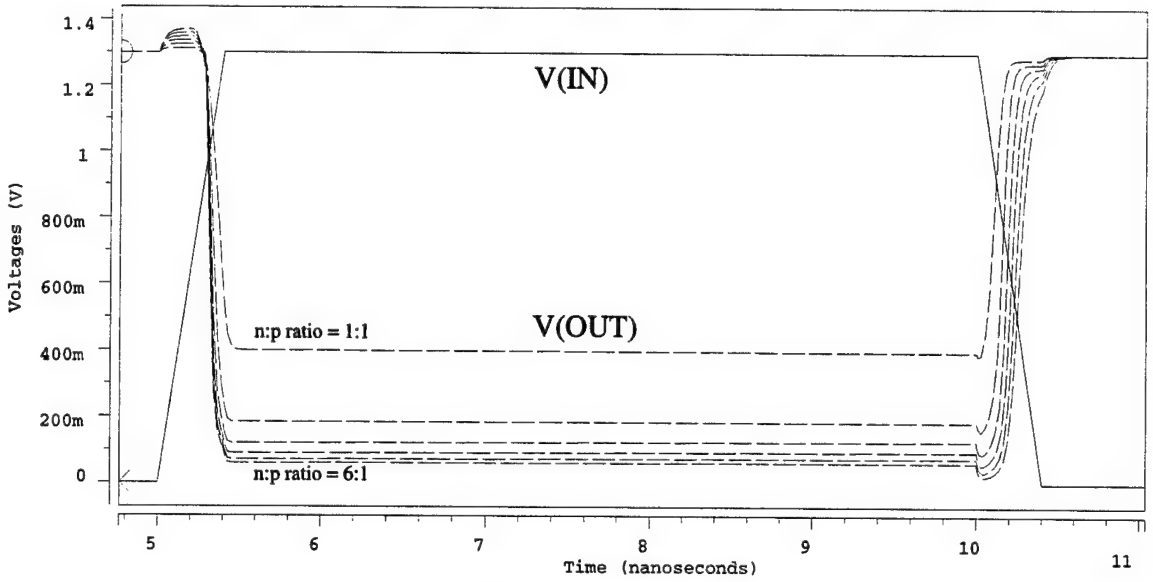


Figure 2.4: CGaAs DCFL NAND2 simulation waveform.

reach an acceptable V_{OL}. As the n-type transistor width is increased, the current through the series n-type transistor network increases, allowing the circuit to achieve a lower V_{OL}. The increased pulldown current also decreases the output fall time and falling propagation delay. At the same time the p-type transistor is now somewhat undersized and the output rise time and rising propagation delay increase. Thus, it becomes difficult to optimize between V_{OL}, rise/fall time, and rising/falling propagation delay, as all of these characteristics are affected by the width of the series n-type transistors. In short, DCFL offers high switching speed at the expense of power and output low voltage, but is cost effective in terms of the transistor count and area required to implement a given function.

2.2.5.2 Complementary Logic

The availability of a p-type transistor enables the CGaAs technology to support fully complementary circuit techniques. The topology of a complementary two-input NAND circuit is shown in Figure 2.5. Complementary circuit techniques overcome many of the disadvantages of DCFL. Complementary logic is a ratioless logic style, meaning that if the n-type and p-type transistors are equally sized, the circuit will function correctly, although not necessarily optimally. Under static conditions, a conducting path exists from the output node to either VDD or GND, not to both, resulting in low static power dissipation.

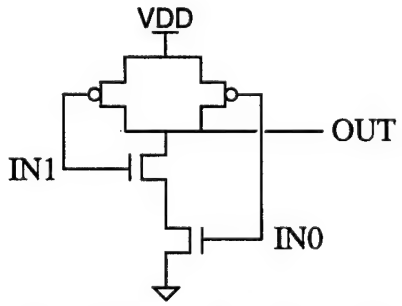


Figure 2.5: Complementary NAND2 circuit implementation.

This also enables complementary logic gates to reach maximum V_{OL} and V_{OH}, resulting in a full rail-to-rail voltage swing with good noise margins.

While complementary circuits offer reasonable switching speeds with fairly low power dissipation, there remains a component of the power dissipation that is of considerable interest. Under transient conditions, there exists a conducting path between VDD and GND, for a brief period of time, while both the n-type and p-type transistors are biased in the saturation region. This results in a current spike, as in fully complementary CMOS circuits, causing the circuit to dissipate a measurable amount of switching power. An HSPICE simulation (see Figure 2.6) demonstrates this. The figure plots the output voltage and switching current versus the input voltage for a series of complementary NAND2 circuits. The n-type and p-type transistor widths are equal, ranging from 0.5 μm to 3 μm, with a channel length of 0.5 μm. The switching currents observed for these devices is on the order of tens of μA, with larger transistors generating proportionately larger currents. The only difference between the behavior of CMOS complementary circuits and CGaAs

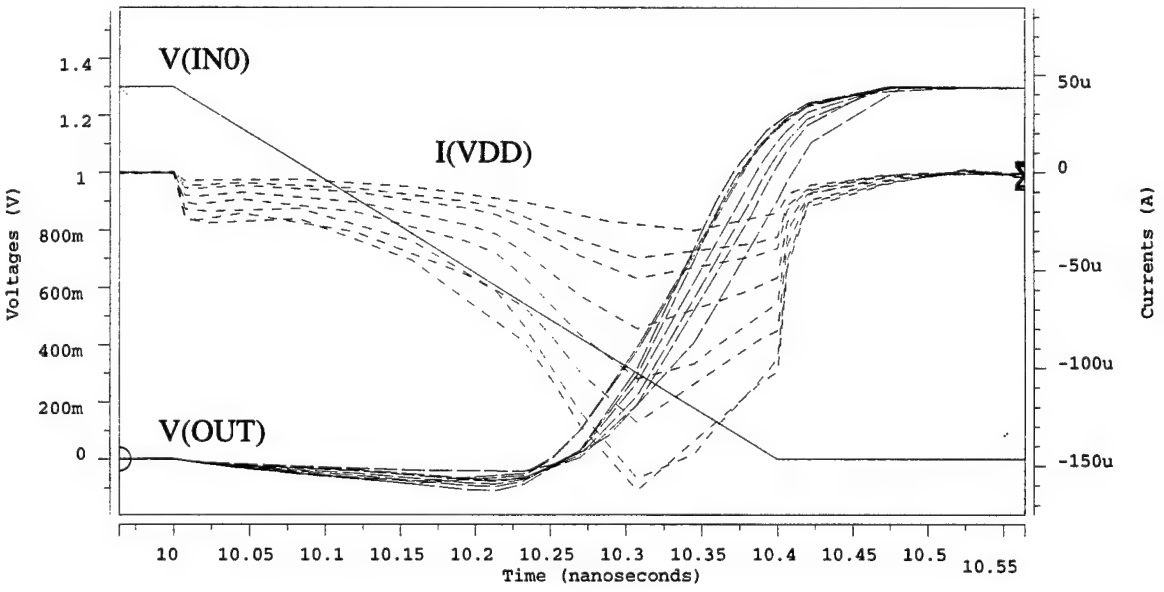


Figure 2.6: CGaAs complementary NAND2 circuit simulation waveform.

complementary circuits is the higher transconductance of CGaAs, resulting in higher power dissipation.

Several other properties of CGaAs render complementary circuits less than ideal. The transconductance of CGaAs p-type transistors is only one-fourth that of the n-type transistors. Thus, for a simple inverter, the p-type transistor must be almost four times larger than the n-type transistor to match output rise and fall times. This increases the input capacitance of complementary logic circuits, which in turn requires transistors to be sized larger still. The result is that the complementary circuits are relatively large, have significant input capacitance, and are slow. A complementary circuit also requires more transistors than a DCFL implementation. Whereas DCFL circuits require only one p-type device to pull the output to a logic high level, complementary circuits require as many p-type transistors as n-type transistors to implement the required functionality. In complex circuits, almost twice as many transistors are required for complementary circuits as for DCFL. These factors result in a considerable area cost for complementary circuits.

2.2.5.3 Dynamic Logic

The disparity between the transconductances of the n-type and p-type transistors leads to complementary logic circuits which are relatively slow. Furthermore, in complementary logic circuits and other static logic families, the voltage level of the output node may transition several times during evaluation, depending upon the input arrival times. Dynamic logic overcomes these inefficiencies by dividing circuit operation into two distinct phases. A capacitive circuit node is charged during the precharge phase, and then conditionally discharged during the evaluation phase. The main advantage of dynamic logic is the fast

switching speed. There is some overhead associated with the precharge phase, during which time no useful work is performed, but this is amortized over all dynamic logic stages in a given circuit. The quick evaluation time of each dynamic gate more than compensates for the overhead associated with the precharge phase.

In the PUMA research project, we investigated the use of domino logic [10], a particular style of dynamic logic. The topology of a domino logic circuit implementing the two-input AND function is shown in Figure 2.7. This type of domino logic circuit is referred to as D1. Figure 2.8 is an HSPICE simulation showing the operation of the Domino circuit. In the precharge phase, the time when the precharge signal, pclk, is asserted low, transistor M3 charges the circuit capacitance C0. The output inverter guarantees that the gate output, and consequently the inputs to other domino gates, will be conditioned to a low voltage level. If IN0 transitions to logic high, charge sharing between C0 and C1 could potentially cause the voltage level of node Z to drop and trigger false discharges of other domino gates. To reduce the effects of charge sharing, all internal nodes are precharged in the same manner as node Z. Transistor M5 precharges the capacitance C1 during the assertion of pclk. Thus, after the precharge phase, all internal nodes are precharged high, and the input to every domino gate is low. Transistor M4 is referred to as a keeper, or

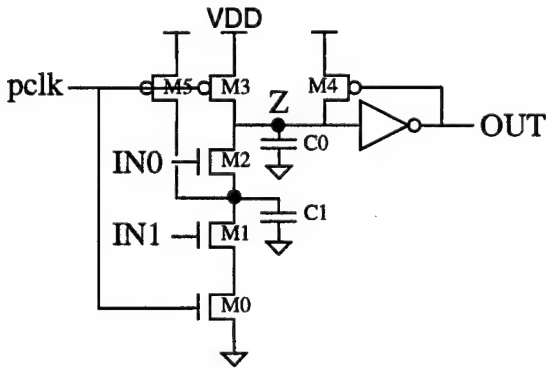


Figure 2.7: Domino logic AND2 circuit implementation.

bleeder, transistor. This device is a small transistor designed to preserve the charge on node Z. When OUT is precharged low, M4 is on, restoring the charge which leaks off of node Z. Without M4, node Z would be much more susceptible to coupled noise, and would not maintain its state if the clock were stopped. The evaluation phase begins with the deassertion of pclk, turning on the evaluation transistor, M0. The inputs to the gate are permitted to have only a rising transition; if both IN0 and IN1 resolve to a logic high level, M1 and M2 will turn on and discharge node Z, which in turn causes OUT to rise.

Because of the overhead of the precharge phase, a single domino logic gate does not provide a significant performance advantage over other logic families. However, a series of domino gates can implement wide, complex functions. Only one short precharge phase will be required to precharge all internal Z nodes to VDD. Once the evaluation phase begins each gate will quickly resolve and conditionally discharge its Z capacitance. In this fashion, the internal nodes in a series of domino logic gates can evaluate and discharge,

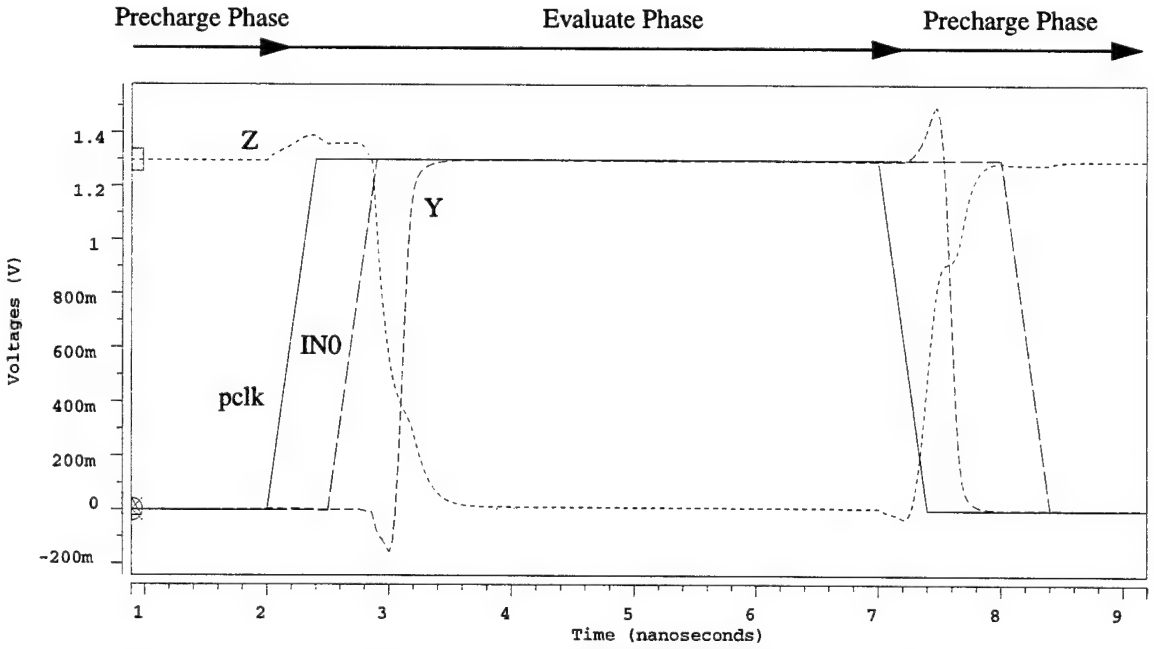


Figure 2.8: Operation of a CGaAs AND2 domino logic circuit.

falling like a series of dominoes. The longer the chain of domino gates, the greater the improvement in delay versus a DCFL or complementary circuit.

This style of domino logic is fairly simple to control. The only requirements are that the precharge signal must have a good slew rate and be asserted long enough to charge all internal nodes to a high voltage level. Domino logic does have several shortcomings. The n-type evaluate transistor adds delay to the discharge path. The precharge phase consumes time and may become a significant overhead for short domino logic paths. The output inverter dictates that only non-inverting logic can be generated and also adds an extra gate delay per stage. These overheads reduce the potential performance gains of domino logic. However, domino circuits allow the designer to implement wide, complex functions in a single domino gate. Thus domino circuits are high-speed, as well as cost-efficient. The only major drawback is the high power dissipation.

2.2.5.4 Summary of CGaAs Digital Logic Families

The low series resistance and availability of a p-type transistor enables a variety of circuit topologies to be implemented with the CGaAs technology. Table 2.3 lists the circuit techniques that were addressed in this chapter and the relative advantages and disadvantages of each. DCFL logic offers fast switching speed and reduced area at the expense of power and degraded noise margins. Domino logic is a cost-effect means of achieving extremely fast gate speed and acceptable noise margins but dissipates the most power. The complementary logic family provides adequate switching speed and excellent noise margins while having fairly low power dissipation, but requires more transistors and area then

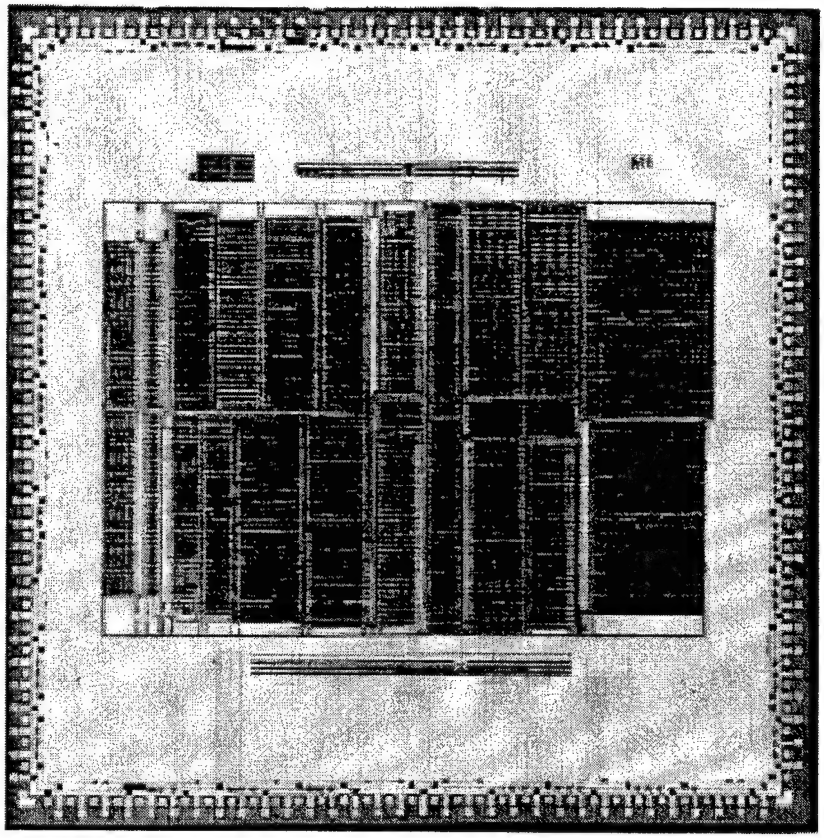
Logic Style	Speed	Power	Area	V_{OL}
DCFL	Good	High	Low	Poor
Complementary	Moderate	Low	Moderate	Good
Domino	Best	High	Low	Good

Table 2.3: Characteristics of CGaAs circuit techniques.

either DCFL or domino circuits. The diversity of the logic families allow IC designers to select a circuit technique that is suitable for a specific application, or to mix circuit styles throughout a design to optimize speed, power, and area.

2.2.6 Design Example: 32-bit Arithmetic Logic Unit

A prototype 32-bit PowerPC Arithmetic Logic Unit (ALU), shown in Figure 2.9, was designed to evaluate the effectiveness of domino logic in the CGaAs technology. To my knowledge, this was the first implementation of domino logic in a complementary GaAs



- PowerPC ALU test chip
- 0.5 µm CGaAs, 3 metal layers
- Domino execution logic
- DCFL & complementary decoders and mux trees
- 102,418 transistors
  5.8 x 5.3 mm chip
  5.5 x 4.2 mm core
- 750 ps 32-bit adder
- 615 MHz
- 3.8 Watts @ 1.5 V

Figure 2.9: Die plot and chip statistics for 32-bit arithmetic logic unit.

process. The execution unit contains an adder, bi-directional shifter, logical unit, and a leading zero counter. The control unit is implemented in a mixed logic style, utilizing both DCFL and complementary logic gates. Negative-edge triggered D flip-flops implement a single-phase clocking scheme. A segmented scan chain, consisting of a full scan chain with multiple input and output ports, ensures testability. The standard cell library, constructed using Mentor Graphics IC Station, consists of domino, complementary, and DCFL logic gates. Domino logic was utilized exclusively in the execution units, where critical paths were longest and the minimum delay was desired. Speed-sensitive control blocks were implemented in DCFL, while non-critical paths were designed with complementary circuits to conserver power. The gates were sized using an optimization approach that minimized the power-delay product, while balancing the rise and fall propagation delays. Cascade Design Automation's Epoch Integrated Circuit Design System provided cell compaction, floor planning, placement, and automated routing. Top level signal routing, power plane construction, LVS and DRC verification were performed within the IC Station framework. The chip integrated 102,418 transistors on a 7.2 x 7.2 mm^2 die, and was to be packaged in a 256 pin TBGA. Based on HSPICE simulations and TACTIC timing analysis, the ALU was expected to operate at 617 MHz, dissipating 3.84 W at 1.5 V.

A number of technical difficulties hindered the effort to package and test the chips. The design was predicated upon aggressive scaling of the transistor threshold voltages, from ±0.55 V to ±0.3 V. The foundry was unable to consistently produce devices with the desired threshold voltages. The fabricated circuits also had a conducting path between the channels of complementary n-type and p-type transistors whose gates were connected by a continuous piece of gate metal. The conduction path was created because the oxygen iso-

lation implant is blocked by the gate metal, leaving a continuous InGaAs channel region (see Fig. 2.1) between the n-type and p-type transistors. Once this was identified, a guideline was established for future designs; the gate metal connecting the gates of complementary n-type and p-type transistors would have to be strapped with a segment of first layer metal, allowing the isolation region to separate the n-type and p-type channel regions, essentially creating an open circuit in the conducting path. A custom packaging solution was developed for chips. The die had two rings of peripheral pads, with the pad locations staggered between the inner and outer rings. The bonding rules were relaxed to allow the chips to be aggressively bonded into a TBGA package. The same vendor was to supply the TBGA packages and perform the bonding. When samples were returned from the foundry, the vendor determined that the packaging solution would not work. The design did not contain test structures to permit testing with only a few probes. It would have cost a great deal of money to test the bare die. Together these technical difficulties prevented us from making further investments in packaging and testing the samples. However, the HSPICE simulations succeeded in demonstrating the performance potential of CGaAs domino logic and the need for lower threshold voltages to design high-speed circuits.

2.3 Summary

The Aurora processors explored the performance potential of GaAs E/D MESFET technology. The high source resistance of GaAs MESFETs limited the use of series enhancement transistors, requiring designers to use NOR-only logic techniques. The Schottky barrier diode between the gate and channel clamps the logic high level to 0.7 V. Circuit designers had to deal with a small logic swing that reduced noise margins; the logic low was 0.1 V and the logic high was 0.7 V. The comparatively large threshold volt-

age, nearly 1/3 of the logic swing, reduced the amount of drain current, thus limiting the switching speed of the transistors.

For the PUMA research project, we chose to experiment with the CGaAs process technology. The CGaAs process offers good power-delay performance and radiation tolerance, making the technology appealing for space and satellite applications [11]. However, CGaAs also presents design challenges such as reduced power-supply voltage, proportionately large device thresholds, gate leakage, and subthreshold drain-source leakage. The most significant design challenge from a system perspective, is the low transistor integration level of the CGaAs technology. The remainder of this dissertation deals with the design, optimization, and implementation of a CGaAs microprocessor designed to maximize performance within the limited transistor budget available in this advanced process.

CHAPTER 3

EVOLUTION OF THE POWERPC ARCHITECTURE

The previous chapter introduced the CGaAs™ technology. The objectives of this research are to evaluate the CGaAs technology for digital VLSI applications, to develop design tools and methods for implementing these circuits, and to demonstrate the capabilities of CGaAs in the design of a CGaAs microprocessor. The biggest design challenge, from an architectural perspective, is the low transistor integration level of CGaAs. Modern general-purpose microprocessors consist of tens of millions of transistors. At the beginning of this research project, the integration level that CGaAs would be able to support in five years was estimated to be one million transistors, but that was only an estimate. The architecture had to be flexible enough to adapt to changing estimates in the level of integration and still offer reasonable performance. In the early phases of this project the PowerPC architecture was being widely promoted in the popular literature. One of the characteristics of the PowerPC architecture was flexibility, demonstrated by the diverse collection of microprocessors that have been implemented to date. These factors, coupled with our relationship with Motorola as a foundry, led to the selection of the PowerPC instruction set architecture as the architectural basis for this research. In this chapter we will review the history of the PowerPC architecture and present the architectural philosophy of our research processor.

3.1 The POWER Architecture

To understand the essence of the PowerPC architecture it is first necessary to study the precursor of the PowerPC architecture, namely IBM's POWER architecture. The POWER (Performance Optimized with Enhanced RISC) architecture was a descendent of the IBM 801 minicomputer. The 801 is believed by some to be the first machine to embody the philosophy of reduced instruction set computing (RISC) [12]. The POWER architecture adheres to the fundamentals of RISC by defining fixed-width instructions, register-to-register operations, and simple addressing modes and instructions. The main objective of the POWER architecture was to further enhance the concepts of RISC, as implemented in the 801 architecture. Specifically, the designers of the POWER architecture wanted to explicitly embody the concepts of superscalar execution in the instruction set architecture (ISA) and make the architecture a better target for compilers [12]. The conceptual model illustrates the attempt to expose superscalar execution in the ISA. The architecture is logically divided into three separate function units: a branch processor, an integer (or fixed-point) processor, and a floating-point processor. The branch processor fetches and dispatches instructions to the fixed-point and floating-point units, while also processing branch instructions. By processing the branch instructions, the branch processor is able to direct a sequential stream of instructions to the fixed-point and floating-point processors. Each processor has its own set of registers; dedicated instructions are used to transfer data from the registers in one processor to those in another. The idea behind this scheme was to expose parallelism among the branch, integer, and floating-point instructions, while minimizing resource sharing and synchronization between the processors [13]. This allows greater parallelism and exposes the individual logical units to the compiler.

While the POWER architecture enhanced the basic fundamentals of RISC architecture, it also diverged from other contemporary RISC implementations. Unlike the MIPS processors, the POWER architecture did not utilize branch- or load-delay slots. IBM recognized that the delay slots would be ineffective in multiple issue machines. Instead, the POWER architecture implemented a fully interlocked pipeline and a large condition register set. The compiler could potentially use the set of condition registers to compute branches early, and direct the instruction stream accordingly. A bit in the instruction format is used by the compiler to indicate instructions which will update the condition register, allowing the compiler to easily reschedule code without destroying branch conditions. The POWER architecture also introduced compound instructions for common instruction sequences. For example, the load-with-update instruction can be used for indexing arrays.

3.1.1 POWER1

In 1990 IBM introduced the RISC System/6000 (RS/6000), the first system based upon the POWER architecture. The processor used in these systems is referred to as POWER1. The POWER1 chip set was a superscalar multichip implementation of the POWER architecture. The processor was implemented in three chips: the instruction cache unit (ICU), fixed-point unit (FXU), and floating-point unit (FPU). Memory and peripheral chips brought the total to nine chips for the entire POWER1 implementation. The machine structure resembled a parallel pipeline with each processor unit organized as a separate pipeline. The ICU pipeline was capable of supplying two instructions per cycle to the FXU and FPU pipelines. The FXU and FPU had four-entry queues at the inputs,

allowing each processor to accept up to two instructions per cycle. The instruction queues made POWER1 a decoupled architecture.

The ICU implemented a two-stage pipeline. In the first stage of the pipeline, up to four instructions were fetched per cycle, while branches were resolved in the second stage. The FXU implemented a four-stage pipeline for integer instructions. The FPU had a six-stage pipeline, with some instructions requiring multiple cycles for the execution stage. Floating-point load and store instructions were executed in both the FXU and FPU pipelines, with the FXU interfacing to the memory system. To account for the potential slip between the two pipelines, the FPU implemented a register renaming scheme. Under the renaming scheme 32 architectural floating-point registers were mapped to 38 physical registers using a map table, free list, and return queue. Six physical registers were available for renames; at any moment these six registers were in either the free list or the return queue.

3.1.2 POWER2

IBM introduced the eight-chip, 23 million-transistor POWER2 in 1993. Multichip module technology was used to integrate the high-end POWER2 processor implementations. POWER2 maintained the same functional partitioning as POWER1, with the processor divided across the ICU, FXU, and FPU chips. POWER2 could issue up to six instructions per cycle; two branch instructions in the ICU, and as many as four integer and floating-point instructions between the FXU and FPU. The resources were essentially doubled over POWER1; each processor had two nearly identical pipelines and the data cache was dual-ported. These attributes, coupled with large, low-latency caches, and high-speed

multichip module (MCM) technology accounted for the high performance of POWER2 systems.

The ICU could fetch up to eight instructions per cycle during the first-stage of the pipeline. In the second-stage, up to two branch instructions could be executed and up to four partially decoded instructions could be issued to the FXU and FPU pipelines. The FXU and FPU both had eight buffers that were capable of accepting up to four instructions per cycle from the ICU. The FXU could read up to two instructions per cycle from these buffers, while the FPU could read up to four instructions per cycle. The number of physical registers was also increased from 38 registers, in POWER1, to 54 registers in POWER2. In addition to the greater rename ability, the FPU could, in some cases, process more than four instructions per cycle. The FXU implemented a three-ported arithmetic logic unit (ALU) that allowed dependent adds to be fused and performed in a single cycle.

POWER2 was a machine of amazing complexity at its inception. Unlike today, performance was not gained through higher clock rate, but instead through increased resources utilized to achieve a high degree of parallelism. The clock rate of the initial POWER1 implementation was 62.5 MHz, compared to 71.5 MHz for POWER2 [13]. The issue rate of POWER2, six instructions per cycle, was the highest among contemporary RISC designs.

3.2 Philosophy of the PowerPC Architecture

In 1991 IBM, Motorola, and Apple formed an alliance with the goal of developing a common RISC architecture that would simultaneously satisfy their individual market requirements. IBM was seeking a successor to POWER2 for their high-end workstations

and servers, Apple was looking for a microprocessor to use in their line of personal computers, and Motorola was seeking an alternative to the 88000 architecture. IBM's successful POWER architecture was chosen as the basis for this new RISC architecture, which was called PowerPC. The goal of the PowerPC development was to enhance the POWER architecture, enabling a variety of single-chip implementations to be developed, ranging from low-cost, to low-power, to high-performance.

The enhancements to the POWER architecture centered around two ideas, simplification and compatibility. The architecture had to be simplified to ease implementation and permit higher operating frequencies. The PowerPC architecture had to be backward compatible with POWER, so that IBM could provide a smooth transition of its existing RS/6000 customer base. A distinction was drawn between architecture and implementation, essentially eliminating the logical partitioning that characterized the POWER architecture. Barriers to high degree (greater than 3) superscalar execution were removed. The subtract-without-carry instruction and multiply-quotient register were removed, eliminating a resource conflict common in the POWER architecture. Instructions such as absolute-value (abs) and negative-absolute-value (nabs), which would potentially impact clock frequency, were eliminated. Additional high-performance features to support symmetric multiprocessing and a 64-bit architecture were defined.

3.3 PowerPC Design Examples

There have been three generations of PowerPC implementations to date. The first generation, the PowerPC 601, was an attempt to bridge the gap between the traditional POWER architecture and future implementations of the PowerPC architecture. The sec-

ond generation of PowerPC processors was dominated by the 603 and 604 implementations. The 603 was targeted as a low-cost, low-power implementation, while the 604 was intended to be a general-purpose desktop microprocessor. The last processor of the second generation was the 620, an ultra-high-performance implementation that did not materialize. The third generation of PowerPC processors, the G3-series, is being led by the 750. A review of these processors demonstrates the flexibility of the PowerPC architecture, allowing a variety of implementations. Some of the processor derivatives focus on low-power and low-cost while others emphasize high-performance.

3.3.1 PowerPC 601

The first implementation of the PowerPC architecture was the PowerPC 601 microprocessor [14]. The objective of the 601 was to establish the PowerPC architecture, while offering competitive performance at low cost. The processor had to be suitable for a wide range of applications. The chip was designed on an aggressive 12 month schedule. The processor was derived from the IBM RISC Single Chip (RSC) Microprocessor [15] and the Motorola 88110 [6]. The logical organization of the 601 was similar to the POWER architecture. The single-chip processor consisted of a Branch Processing Unit (BPU), a Fixed-Point Unit (FXU), and a Floating-Point Unit (FPU). The 601 could dispatch up to three instructions per cycle, one to each of three functional units. An on-chip 32 KB unified cache stored both instructions and data. The 601 did not implement register renaming, except in the BPU where the Link Register (LR) and Count Register (CTR) were renamed. The renaming allowed the BPU to speculatively execute branch instructions with respect to the FXU. The BPU implemented a static branch prediction scheme. Backward branches

were predicted taken, while forward branches were predicted not taken. This allowed the 601 to implement loops efficiently. The compiler could set a bit in the instruction word to reverse the static prediction. Instructions following the predicted branch could be dispatched, but execution was delayed until the prediction was resolved. An out-of-order dispatch scheme, based on a tagging and counting mechanism to maintain program order, allowed subsequent branches to be uncovered, while FXU instructions were interlocked. Unlike previous POWER implementations, and subsequent PowerPC implementations, the 601 utilized a load-delay slot to simplify dependencies. The 601 was implemented in 0.6 μm, 4-layer metal CMOS. The 2.8 million transistor 601 had a die size of 10.95 x 10.95 mm. The 601 operated from a 3.6 V supply, dissipating 6.5 W at 50 MHz.

3.3.2 PowerPC 603

The PowerPC 603 processor [16] was the first PowerPC microprocessor to be optimized for low-power and low-cost applications. The 32-bit superscalar microprocessor was designed in a 0.5 μm, 4-layer metal CMOS process. The 7.4 x 11.5 mm die integrated 1.6 million transistors. The initial implementation ran at 80 MHz and consumed only 2.5 W, while achieving 85 SPECint92 and 85 SPECfp92 [17]. The microarchitecture was much different than the 601 or POWER processors. The 603 had five execution units; branch, integer, floating-point, load-store, and system. The 603 was capable of issuing three instructions per clock cycle. The processor implemented an in-order dispatch policy while issuing and executing instructions out-of-order. The 603 implemented distributed reservation stations and register renaming. Like the 601, the 603 implemented a simple static branch prediction scheme, allowing speculation past unresolved branches. The

instruction and data caches were each 8 KB, 2-way set associative. The processor implemented the 60X bus, a 64-bit, pipelined bus capable of supporting split-transactions. The 60X bus was implemented in both the 603 and 604 processors.

3.3.3 PowerPC 604

The PowerPC 604 microprocessor [18] was intended to be a general-purpose desktop microprocessor. The processor had 16 KB instruction and data caches. The 32-bit superscalar 604 could dispatch up to four instructions per cycle. The 604 had six execution units; two single-cycle integer, one multicycle integer, load-store, floating-point, and branch. The 604 had an in-order dispatch scheme, with out-of-order reservation stations. Register renaming was accomplished via a 12-entry rename buffer for integer registers and separate rename buffers for floating-point registers, and the condition register. A 512-entry branch history table (BHT), comprised of two-bit counters, performed dynamic branch prediction, an improvement over the static scheme utilized in the 601 and 603 processors. The 604 could speculatively fetch and execute instructions beyond two unresolved branches. The 3.6 million transistor 604 was designed in a 0.5 μm, 4-layer metal CMOS process. The 12.4 x 15.8 mm 604 consumed 10 W at 100 Mhz. The processor conformed to the 60X bus specification.

3.3.4 PowerPC 620

The PowerPC 620 microprocessor [19] was the first 64-bit implementation of the PowerPC architecture. The processor was intended to be an ultra-high performance PowerPC processor, targeted at high-end workstation and server applications. The instruction and data caches were 32 KB, 8-way semi-associative, with 64 B line size. The semi-associa-

tive policy divided the cache into sixty-four eight-entry content addressable sectors. This scheme was designed to conserve power and improve speed over a true 8-way set-associative policy. Predecode bits were stored in the instruction cache. The 620 could sustain a maximum execution rate of 4 instructions per cycle. The machine structure was similar to the 604, consisting of the same six execution units. The in-order dispatch, out-of-order execution, in-order completion scheme was implemented with a 16-entry reorder buffer. The 620 could speculatively execute instructions past as many as three unresolved branches. Branch prediction resources were also increased over those of the 604. The 620 used a 2048-entry BHT, a call and return stack, and a 256-entry 2-way set-associative branch target address cache to predict branches and target addresses. The branch unit also contained a shadow count register to detect the end of loops. The 620 implemented a new high-performance processor bus, capable of supporting 8-way multi-processing. The 66 MHz bus had a peak bandwidth of 1 GB/sec. The 620 was to be implemented in 0.5 μm, 4-layer CMOS. The 620 would have required an estimated 7 million transistors, and have had a die size of 17.1 x 18.2 mm. The processor dissipated an estimated 30 W at 133 MHz. Due to the complexity involved, the design schedule slipped and the 620 never reached the market place.

3.3.5 G3-series: PowerPC 750

The third generation of PowerPC microprocessors represented a change from the trend seen in the 6XX series. The second generation of PowerPC processors was achieving high performance through a high degree of parallelism at relatively low frequencies. This led to the immense complexity, and ultimately the failure, of the 620. The third generation,

dubbed the G3-series, scaled back the complexity and opted to exploit higher frequencies. The PowerPC 750 [20,21] was able to achieve high-performance, 10 SPECint95, with surprisingly low-power, 5 W at 250 MHz. Most of the transistors, 90% of the 6.35 million transistors, were allocated to the implementation of separate 32 KB, 8-way set-associative, instruction and data caches. Branch prediction was accomplished via a 512-entry BHT. The 750 would fetch past two unresolved branches, but would only speculate past one unresolved branch. The 750 consisted of five functional units; Complex Integer Unit (CIU) for multi-cycle multiply and divide operations, Simple Integer Unit (SIU), System Unit (SYS), Load-Store Unit (LSU), and floating-point unit (FPU).The superscalar execution core was capable of issuing two instructions per cycle and supported out-of-order execution with only 6 integer and 6 floating-point rename registers. In fact, most of the functional units had only a single reservation station. This simple superscalar design approach was a surprising departure from contemporary microprocessor designs. The chip was fabricated in a 0.25 μm, 5-layer metal CMOS. The small die size, only 7.6 x 8.8 mm, low power, and reasonable performance has made the 750 a popular microprocessor for desktop and notebook applications.

3.4 Summary of PowerPC and POWER

This review of the PowerPC architecture has examined the underlying philosophy and demonstrated the flexibility of the PowerPC architecture. The POWER architecture, the predecessor of the PowerPC architecture, was an extension of the 801 minicomputer. The goal of the POWER architecture was to expose the superscalar nature of the machine to the compiler. To this end, the POWER architecture identified three processing elements; the branch processor, the fixed-point processor, and the floating-point processor. Each pro-

cessing element had its own instructions and register set. The POWER architecture attempted to minimize communication and synchronization between the processors, achieving a high degree of parallel execution. The PowerPC architecture is essentially a redefinition of the POWER architecture. The POWER architecture was enhanced to permit high degree superscalar execution, while supporting a variety of single-chip implementations, ranging from low-cost to high-performance.

The progression of the PowerPC architecture can be traced from the original POWER1 processor through the current PowerPC 750 microprocessor. A summary of the POWER and PowerPC processors discussed in this chapter is presented in Table 3.1. POWER1 was a fairly straightforward implementation of POWER, while POWER2 essentially doubled the resources to achieve higher performance. Since the POWER1 and POWER2 implementations spanned multiple chips, it is difficult to obtain accurate area, power, and transistor estimates, some entries in Table 3.1 are therefore left blank. The PowerPC 601 was a simple implementation of the PowerPC architecture. The 601 resembled a single-chip implementation of the POWER1. The PowerPC 603 and PowerPC 604 were the first orig-

Processor	Year	Frequency (MHz)	SPECint		Cache (KB)	Units /Issue	Tech. (μm)	Size (mm^2)	Power (W)	Transistors (10^6)
			92	95						
POWER1	90	62.5	na	na	72	2/2				
POWER2	93	71.5	na	na	288	4/6				23.0
601	93	50	40	na	16	4/3	0.60	121	6.5	2.8
603	94	80	75	na	16	4/2	0.50	50	2.2	1.6
604	94	100	128	3.6	32	6/4	0.50	197	14.0	3.6
620	95	133	225	6.0	64	6/4	0.50	311	30.0	7.0
750	97	250	na	11.8	64	6/3	0.25	67	5.0	6.4

Table 3.1: Summary of POWER and PowerPC microprocessors.

inal PowerPC designs, only vaguely resembling the POWER implementations. The 603 was a low-cost, low-power implementation, while the 604 was a high-performance desktop processor. The PowerPC 620 was similar to the POWER2 in that it attempted to improve performance by increasing the on-chip resources, such as functional units and caches, instead of improving frequency. The approach that was taken with the PowerPC 620 epitomized the battle of "the brainiacs vs. the speed demons" [22], advocating higher performance through increased parallelism, rather than through increased clock frequency. The failure of the 620 may have led to the less complex and faster PowerPC 750. The 750 used a fairly simple superscalar core, consisting of only six integer rename registers and a single reservation station per functional unit, along with an increased clock frequency to achieve higher performance. These design examples illustrate the flexibility of the PowerPC architecture.

3.5 Architectural Philosophy of the CGaAs Microprocessor

The PUMA project is an investigation into the use of advanced technologies. The CGaAs and MCM technologies, along with the PowerPC architecture form the basis of the research project. The remainder of this dissertation deals with the optimization and design of the fixed-point unit (FXU), central to the PUMA system. The greatest design challenge is the low transistor integration level of the CGaAs process. Throughout the research project, the CGaAs process was in a constant state of modification and enhancement. The estimated integration level ranged between 1 million to 300,000 transistors at various times throughout the project lifetime. The goal for the CGaAs microprocessor operating frequency was initially targeted at 1000 MHz, this was revised following the CGaAs ALU debacle (see Section 2.2.6). The revised approach emphasized the radiation tolerance and

low-power characteristics of the CGaAs technology, while making increased clock frequency a secondary objective. The flexibility of the PowerPC architecture allowed the microarchitecture of the FXU to adapt to the changing face of CGaAs.

With the initial goal of developing a 1 GHz CGaAs PowerPC MCM system, the processor architecture was to follow the "speed demon" approach. The original FXU architecture implemented a 12-stage superpipeline. But, the inability of the CGaAs technology to support high operating frequencies led to a revised philosophy of the microarchitecture. Since the performance of the system could not be improved through increased clock frequency, the architecture was redesigned to achieve a higher degree of parallelism. Thus, in some sense, the CGaAs microprocessor moved from the "speed demon" camp to the "brainiac" camp.

The transistor budget of the CGaAs FXU can support a number of features implemented in the lineage of PowerPC microprocessors. The principles of superscalar execution embodied in the 750 can be applied to the CGaAs FXU. The 750 used a very limited number of rename registers and reservation stations to support multiple functional units. The CGaAs FXU can improve upon the simple branch prediction strategies implemented in the PowerPC microprocessors. The 601 and 603 used only static prediction. The 604 and 750 microprocessors implemented a 512-entry one-level dynamic scheme, while the proposed 620 architecture called for 2048 entries. These small one-level dynamic branch predictors can be converted to two-level schemes with the addition of an inexpensive branch history register. The CGaAs FXU could potentially benefit from a small two-level dynamic branch predictor. The low integration level of the CGaAs process may not permit

an on-chip data cache. If this is the case, the CGaAs FXU architecture must be able to tolerate the additional latency of an off-chip data cache. The load-delay slot, utilized in traditional RISC designs, would be ineffective in alleviating a multi-cycle data cache latency. The PowerPC architecture does not rely upon delay slots. The FXU architecture will rely upon superscalar execution and branch prediction to overcome the performance limitations imposed by the low integration levels of the CGaAs process.

CHAPTER 4

PUMA SYSTEM OVERVIEW

The goal of the PUMA research project is to demonstrate the performance potential of advanced technologies [23]. This chapter describes the PUMA system. The advanced MCM technology is utilized to overcome the disadvantages of CGaAs by providing a high-speed substrate to interconnect the processor chips. The PUMA programming model is derived from the PowerPC Instruction Set Architecture (ISA). With the description of the PUMA system architecture, the foundation will exist for the subsequent development of the CGaAs PowerPC Microprocessor.

4.1 PUMA System Architecture

The architecture of the PUMA system is shown graphically in Figure 4.1. The processor is divided across multiple chips, conforming to the low transistor integration level of the CGaAs technology. The proposed PUMA processing core calls for an integer processor, or Fixed-Point Unit (FXU), and separate Memory Management Units (MMUs) for instruction and data memory (IMMU and DMMU). The original proposal also detailed a Floating-Point Unit (FPU) that would interface to the FXU. With the FPU, the proposed MCM system would have resembled the POWER2 implementation. However, personnel constraints did not allow the design and implementation of the FPU. The secondary cache

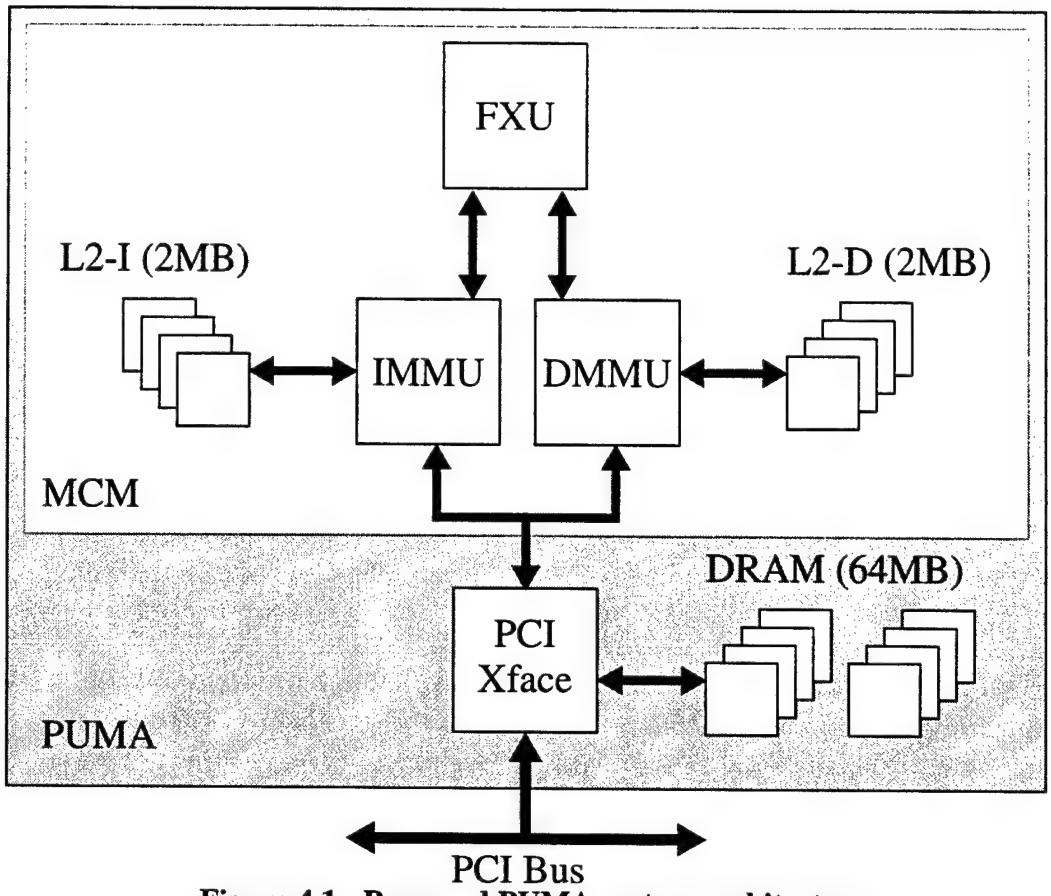


Figure 4.1: Proposed PUMA system architecture.

will consist of dual 2 MB instruction and data caches, implemented with commercial high-speed SRAMs. The processor and SRAM chips are to be assembled onto an MCM using an aggressive flip-chip area-interconnect approach. The MCM, 64 MB of DRAM, and a PCI interface chip are to be mounted on a printed circuit board (PCB) configured to interface with the PCI local bus of a personal computer. To simplify the system design, the PCI interface chip will possess slave-only capability. The host system will be responsible for loading program and data images into the DRAM and initiating PUMA processing. The host system will verify results after the task has been completed.

4.1.1 MCM Advantages

Multi-chip modules offer high-performance alternatives for microprocessor-based systems [24] by enabling inexpensive, high-bandwidth interconnections between ICs. The proposed PUMA system architecture exploits this characteristic by dedicating separate bus structures for instruction and data communication with the secondary cache system. The dual bus architecture will allow for both instruction and data accesses to proceed independently of one another, eliminating interference between the two streams. The processor's instruction fetch mechanism will be able to aggressively prefetch instructions, while the execution core will be free to speculatively access data memory. The width of each bus, 128-bits, will be sufficient to transfer an entire cache line in a single cycle. In addition to the increased width, the bus architecture will implement a split-transaction protocol, allowing up to eight outstanding pipelined requests to be completed out of order. The system bus architecture will be capable of transferring data at a peak rate of 3.2 GB/s when operating at 100 MHz.

While the MCM will provide advantages for system architectures, it will also alleviate some of the shortcomings of CGaAs. The low transistor integration level of the CGaAs technology precludes an on-chip primary data cache. The MCM technology will allow the primary data cache to be located off-chip by providing a high bandwidth connection to the cache. To reduce the delay between the processor and primary cache, these chips will be located in close proximity on the MCM. The 200 MHz pipelined 128-bit primary data cache bus will be able to attain a sustained bandwidth of 3.2 GB/s.

4.1.2 Memory Hierarchy

The memory hierarchy of the PUMA system will be organized into three levels. Like traditional memory systems, each level of the proposed PUMA memory system is smaller, faster, and more expensive than the level below [25]. The MCM, by virtue of the high bandwidth it affords, will effectively reduce the penalty of off-chip memory accesses, compensating, to some extent, for the inability of CGaAs to provide the processor with large on-chip memory structures.

The design of the PUMA primary cache configuration will be influenced by the low integration level of the CGaAs technology. Ideally, the primary cache would be as large as possible in order to reduce the miss rate, though too large a cache would impact the operating frequency due to the longer access times. Architectural innovations might be able to alleviate the effects of the increased miss rate that a smaller cache would incur. The Harvard Architecture of the primary cache has several advantages. Each cache can be tuned to the correct size, line size, and associativity, while eliminating interference between instruction and data memory. The instruction and data caches can be implemented on different ICs, allowing both to be larger in size. This is made possible because of the low-cost high-bandwidth interconnection provided by the MCM.

Like the primary cache, the secondary cache will also be organized in a Harvard Architecture, with a bank of memory for the instruction cache and a separate memory structure for the data cache. Each cache will be connected to the processor through a dedicated system bus and controlled by a memory management unit (IMMU or DMMU). The

size of the secondary cache will be determined by the technology of the MCM and the availability of commercial CMOS SRAM components.

Memory requests that miss both the primary and secondary caches must be satisfied from the physical memory present on the system card. A PCI interface chip will control the 64 MB of DRAM memory mounted on the PCB and communicate with the host system memory through the PCI bus. The MMUs will interface directly with the PCI chip to access physical memory. The host operating system will be responsible for preloading data into PUMA's physical memory space.

4.2 PUMA Instruction Set Architecture

The primary components of the PowerPC Instruction Set Architecture (ISA) [26] are the registers, memory model, exception model, and assembly instructions. Several modifications were made to the PowerPC ISA to simplify implementation, while satisfying the needs of the proposed PUMA system design.

4.2.1 Assembly Instructions

The PUMA assembly-level instruction set is summarized in Table 4.1. The instruction set consists of 75 PowerPC instructions; variations of these instructions bring the instruction count to 133. Most integer arithmetic instructions are supported: add, subtract, shift, rotate, logical and compare. All PowerPC branch instructions are included. Load and store byte, halfword, and word instructions are implemented. The addressing modes for mem-

Functional Category	Instructions	Functions
Integer Arithmetic	add, addc, adde, addi, addic, addic., addme, addze, neg, subf, subfc, subfe, subfic, subfme, subfze	add (carrying, extended, immediate, to negative one, to zero), negate, subtract (carrying, extended, immediate, from negative one, from zero)
Integer Compare	cmp, cmpi, cmpl, cmpli	signed and unsigned comparison
Integer Logical	and, andc, andi, eqv, extsb, extsh, nand, nor, or, orc, ori, xor, xori	(a & b), (a & ~b), ~(a ^ b), sign extend, ~(a & b), ~(a | b), (a | b), (a | ~b), (a ^ b)
Integer Rotate	rlwimi, rlwinm, rlwnm	rotate left then mask insert (or AND with mask)
Integer Shift	slw, sraw, srawi, srw	shift left or right logical, shift right algebraic
Integer Load	lbz, lbzu, lbzux, lbzx, lhz, lhzu, lhzux, lhzx, lwz, lwzu, lwzux, lwzx	load byte, halfword, or word (and base register update)
Integer Store	stb, stbu, sth, sthu, stw, stwu	store byte, halfword, or word (and base register update)
Synchronization	isync, sync	instruction and memory synchronization
Branch	b, bc, bcctr, bclr	unconditional and conditional branch (and link)
System Linkage	rfi, sc	return and call
Processor Control	mcrf, mcrxr, mfcr, mfmsr, mfspr, mtcrf, mtmsr, mtspr	move to and from special registers
Cache Management	dcbf, dcbt	flush and touch

Table 4.1: PUMA assembly instructions.

ory accesses are absolute, register-relative, and register-indexed. Load and store with update instructions are also implemented to allow the software to access sequential memory locations without explicitly incrementing the base address register. Data cache management instructions allow software to prefetch and post-store data from, and to, memory. The additional complexity of supporting memory update instructions and counter-based conditional branches is justified by the performance improvements these powerful constructs provide [27].

Floating-point, integer multiply, integer divide, and 64-bit operations are not supported because of the limited integration level. Load-store-multiple and string instructions are also not implemented. PowerPC instructions not specified in the PUMA ISA can be emulated through software routines.

4.2.2 Register Set

The PUMA register set has been designed to accommodate a variety of general-purpose applications and a simple operating system. It consists primarily of thirty-two 32-bit General Purpose Registers (GPRs) necessary for integer computation, and several special-purpose user-level registers. The PowerPC Condition Register (CR) consists of eight 4-bit condition code fields, set to reflect whether the result of an arithmetic or compare operation is greater than, less than, or equal to zero. The PUMA ISA specifies a CR consisting of only one condition code field. Hall and O'Brien have shown that providing more than a single set of condition codes does not significantly improve performance of general-purpose applications [27]. The Exception Register (XER) has been reduced to only the uppermost bits, storing carry out and overflow status; the lower bits are defined for PowerPC instructions not included in the PUMA ISA. The Link Register (LR) and Count Register (CTR) are used for branch instructions. The CTR holds a count value for loop iterations and the LR stores the return address for subroutine calls. To support operating system tasks, the PUMA ISA defines several supervisor-level registers. The 32-bit Machine State Register (MSR) specifies the operating mode. The Decrement register (DEC) functions as a timer, derived from the core clock frequency. When the DEC is decremented through zero the processor may (depending on the operating mode) signal an exception, transfer-

ring control to the operating system. Several other registers are used to implement precise interrupts: Storage Description Register (SDR1), Data Address Register (DAR), Data Storage Interrupt Status Register (DSISR), and Machine Status Save/Restore Registers (SRRO and SRR1).

4.2.3 Memory Model

The PowerPC architecture defines a 52-bit (4 PB, four petabyte) virtual address space. This is much larger than the amount of memory supported by the proposed PUMA system. To simplify system design, the proposed system will implement only 64 MB of memory on the system card. Given this constraint, virtual memory (VM) was removed from the ISA. Instead, the ISA defines a flat 32-bit (4 GB) physical address space. However, in the proposed system, the upper 6-bits of the physical address will be ignored by the memory subsystem. Removing virtual memory from the ISA eliminates the need for segment registers, translation lookaside buffers, page tables, operating system support, and memory protection, as well as the assembly instructions for manipulating these resources.

4.2.4 Exception Model

The PUMA ISA defines three types of exceptions: asynchronous nonmaskable, asynchronous maskable and synchronous precise. The only PowerPC interrupts not included in the PUMA ISA are those pertaining to instruction and data storage protection and floating-point operation. These exception types were eliminated with the removal of virtual memory and floating-point support.

The PUMA ISA defines asynchronous, nonmaskable hard and soft system resets. The external interrupt and the decrement exceptions are asynchronous and may be masked (by clearing a bit in the MSR). Synchronous precise exceptions, or program exceptions, are caused by the execution of an instruction. The PUMA ISA defines program exceptions for illegal or privileged instructions, providing a mechanism for system software to perform the functions of unimplemented PowerPC instructions such as floating-point computations or integer multiply and divide. The system call exception allows system intervention to perform specialized tasks. The trace exception provides advanced hardware and software debug capabilities. In trace mode, the PUMA processor will single-step through a program, taking an exception after every instruction. Alternately, tracing may be restricted to branch instructions (by specifying a different mode in the MSR). Every exception is fully recoverable, except of course for hard system resets.

4.3 Summary

The proposed PUMA processor consists of several ICs, interconnected by an MCM substrate, enabling high-speed interconnection of the processor chips. The MCM provides high-bandwidth interconnections between ICs, allowing a great deal of flexibility in implementing the PUMA system. A memory hierarchy, based upon the advantages afforded by the MCM technology, was described. The PUMA ISA was defined with the intention of enabling a relatively simple PowerPC implementation, while also maintaining some degree of compatibility with commercial PowerPC microprocessors. The ISA defines the assembly instructions, architectural registers, memory model, and exception model that the PUMA system will support. Most of the ISA will be implemented solely in

the FXU, central to the PUMA system. The remainder of this dissertation is devoted to the design, optimization, and implementation of the CGaAs microprocessor.

CHAPTER 5

MICROARCHITECTURE OPTIMIZATIONS

The architectural studies described in this chapter were performed in the early development stages of the PUMA CGaAs PowerPC^{TM} microprocessor. From an architectural perspective, the greatest barrier presented by the CGaAs technology is the low integration level. At the inception of the PUMA research project, it was forecasted that the CGaAs process would be capable of supporting a one million transistor microprocessor. However, over the course of the project, revised estimates substantially reduced the CGaAs integration level. As Chapter 7 will describe, the CGaAs PowerPC^{TM} microprocessor that was finally implemented consisted of only 383,000 transistors, which was considered aggressive at the time.

In the early phases of the project, it became apparent that the processor would have to be partitioned over multiple chips to compensate for the low CGaAs integration levels. MCM technology provides a virtually unlimited number of fast interconnections between chips, allowing high-bandwidth connections and low-latency signalling [24]. This will alleviate the performance impact of the additional latency penalties associated with a multi-chip processor design. In the PUMA system, MCM technology will be used to provide high bandwidth connections between the processor chip, the off-chip primary data cache, and the secondary caches.

The studies described in this chapter evaluate various aspects of the microarchitecture. The objective of these studies is to optimize the performance of a microarchitecture requiring fewer than one-million transistors. Instruction and data cache performance will be studied in an attempt to optimize the performance of small cache structures. The fundamental performance of superscalar execution and branch prediction will be explored on a small-scale. The concept of the unit-operation will be introduced as an attempt to simplify the implementation of compound PowerPC™ instructions. Finally, several microprocessor configurations will be proposed and the overall performance weighed against the estimated transistor counts.

5.1 Constructing a Baseline Microarchitecture

To evaluate different architectures and performance enhancing features, it is necessary to first establish a baseline model, or reference platform, from which to make improvements. The classic model of a RISC processor is the five-stage pipeline of the MIPS R2000, however we will employ a more modern configuration from which to construct our baseline microarchitecture. A generic superscalar microprocessor similar to the one described by Johnson [28] will serve as the baseline model. The performance characteristics of the instruction and data caches and the superscalar execution core will be evaluated, as well as the effects of other performance enhancements such as branch prediction and instruction prefetching. Comparisons will be drawn between a sequential pipelined microprocessor and the baseline microprocessor configuration.

The baseline microprocessor configuration is shown in Figure 5.1. The processor is two-way superscalar, with on-chip 4 KB instruction and data caches. The machine does

not prefetch instructions or predict branches; performance improvement is based solely on the ability of the superscalar execution engine to extract parallelism from the instruction stream. The integer unit (ALU) and branch unit (BRU) are single-cycle units, while the load (Load) and store (Store) units are pipelined according to the primary data cache latency. In cases where multiple cycles are required to access the primary data cache, the load and store units are configured for pipelined memory accesses, maintaining a throughput of one instruction per cycle. The decode unit is capable of decoding and dispatching two instructions per cycle. Two integer results and one floating-point result may be broad-

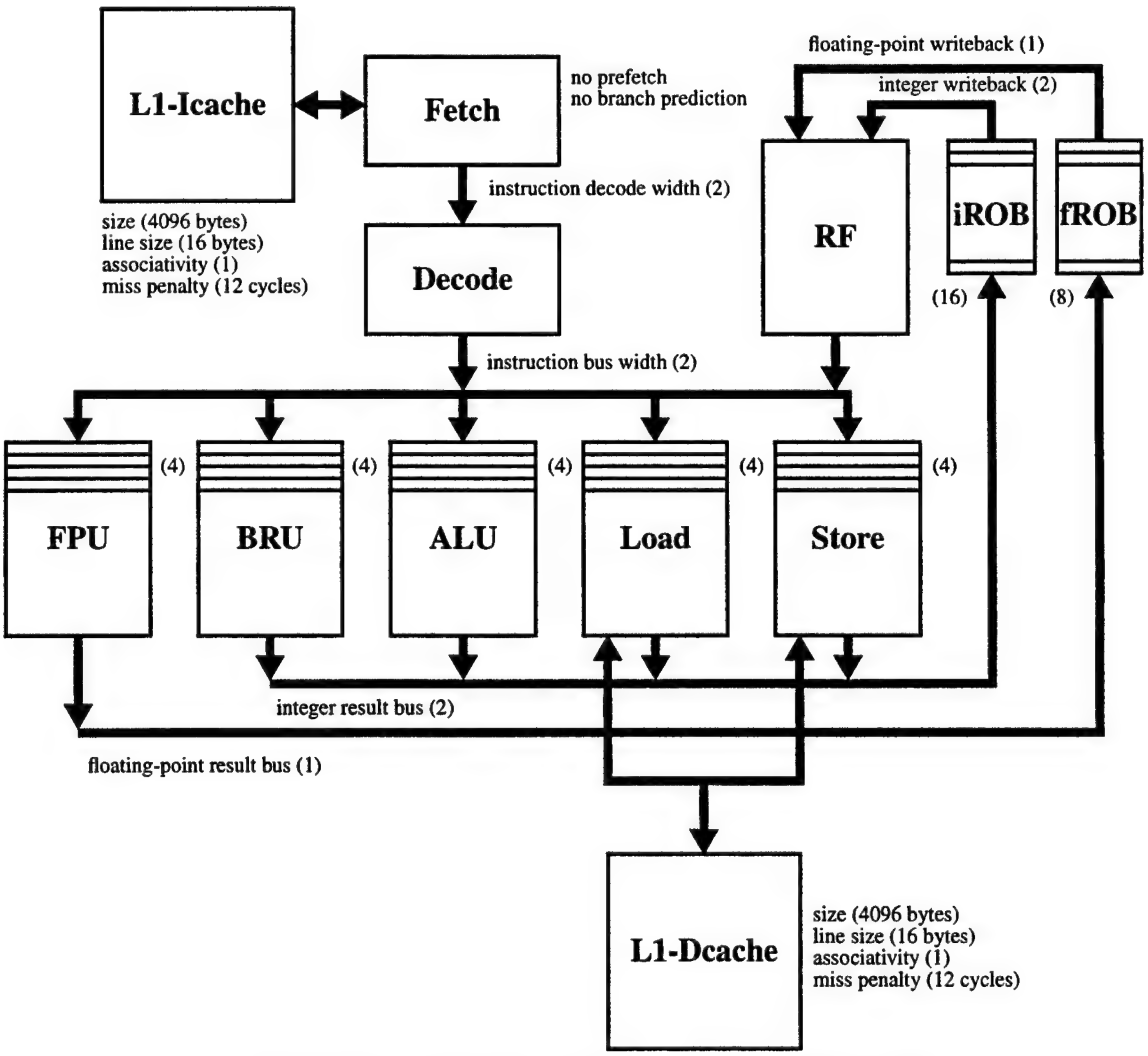


Figure 5.1: Baseline microprocessor configuration.

cast per cycle, while another set of two integer results and one floating-point result may be written back per cycle. The microprocessor is designed to sustain an execution rate of two instructions per cycle (IPC). The instruction window is maintained by a sixteen-entry integer reorder buffer (iROB) and an eight-entry floating-point reorder buffer (fROB), and dependencies are resolved among distributed reservation stations, with four allocated per functional unit. This study is focused on the integer performance of the processor. Design and optimization of the FPU is not explored. Rather, for the sake of completeness in the simulations, the FPU is modeled using conservative instruction latencies. Simulations in the following sections will be based upon this baseline microprocessor configuration. The default parameters for each simulation will conform to the description in Figure 5.1, unless otherwise indicated. The exception to this is the simulation of the various branch prediction schemes described in Section 5.3.6, which for computational limitations were performed with a customized branch-level simulator.

5.2 Simulation Methodology

A simulation environment was developed to facilitate rapid evaluation of architectural features. Johnson's superscalar simulator, ssim [28], was used as the basis for this cycle-level simulator. Extensive modifications were necessary to convert the simulator from MIPS to PowerPC™. Additional improvements were made to the simulator, such as adding instruction prefetch capabilities and a generic programmable branch prediction accuracy. Rather than actually simulating a specific branch predictor configuration, the simulator can be programmed to produce a given percentage of correctly predicted branch outcomes based upon a random number generator. The simulator processes a dynamic program trace and produces a detailed analysis of the performance of various components

of the microarchitecture. A PowerPC^{TM} instruction-level simulator, psim [29], generates the instruction traces.

The Spec95 integer benchmark suite was used as the basis for evaluating performance. Most of the Spec95 programs are fairly small by commercial standards, but the processor we are developing is not intended to compete with commercial desktop processors. Rather, the machine is to function as an embedded special-purpose processor; most of the applications will be small hand-coded programs. The Spec95 programs compress, go, ijpeg, li, m88ksim, and vortex were simulated for 10 M dynamic instructions. The instruction-level simulator used for trace generation could not correctly process gcc and perl, so instruction traces were not generated for those benchmarks.

5.3 Simulation Results

Key features of the microarchitecture were simulated independently to determine how they affected the performance of the baseline model; the metric for comparison is execution rate, as measured in instructions per cycle (IPC). Only enhancements that could be implemented with a relatively small number of transistors were considered. It is obvious that a bigger cache or a second integer unit would offer superior performance over the baseline machine, but these features would consume a significant amount of the already limited on-chip resources. The following sections summarize key elements of the microarchitecture and the techniques available to improve performance. Finally, the various features and enhancements are combined, allowing a comparison of the cost and performance of several different processor configurations.

5.3.1 Instruction Cache Configuration

Processor performance can be improved by taking advantage of the principles of locality that govern memory accesses. Temporal locality dictates that pieces of data will often be used again relatively soon; thus, the need for a cache. Generally, a cache should be as large as possible without impacting operating frequency or yield. The principle of spatial locality implies that the processor tends to access data items that are near to one another in the memory space. Typically, the processor requests an instruction from the primary instruction cache; initially, this generates a cache miss, which causes the instruction to be loaded into the cache. Successive requests are typically for sequential instructions. Spatial locality is the reason that a cache line consists of not one, but multiple instructions. When an instruction misses the cache, the entire line is loaded, providing the processor with several additional instructions to execute. More often than not, the entire cache line is used. Beyond the cache line, spatial locality predicts that if the processor executes instructions in one cache line, it is likely to need instructions from the next cache line shortly. The impact of instruction cache misses can be alleviated by actually fetching sequential lines before instruction cache misses occur. This technique is referred to as instruction prefetching. In this section we explore the basic instruction cache configuration and investigate the benefits of instruction prefetching.

5.3.1.1 Instruction Cache Size

The direct-mapped instruction cache in the baseline machine was varied from 512 B to 32 KB to explore the effects of cache size. A plot of overall processor performance, as measured in instructions per cycle (IPC), versus instruction cache size is shown in Figure 5.2. Performance improves rapidly from 512 B to 8 KB and then begins to level off. The largest performance gains are obtained by increasing the cache size from 2 KB to 4 KB and from 4 KB to 8 KB. The simulation results also demonstrate the effectiveness of a small instruction cache for the SPEC95 benchmarks. Most of the benchmarks have hit rates of 98-99% for an 8 KB instruction cache. While such small application footprints may not be indicative of the workload for a general-purpose desktop microprocessor, the applications that may be run on a special-purpose processor will most likely be fairly small.

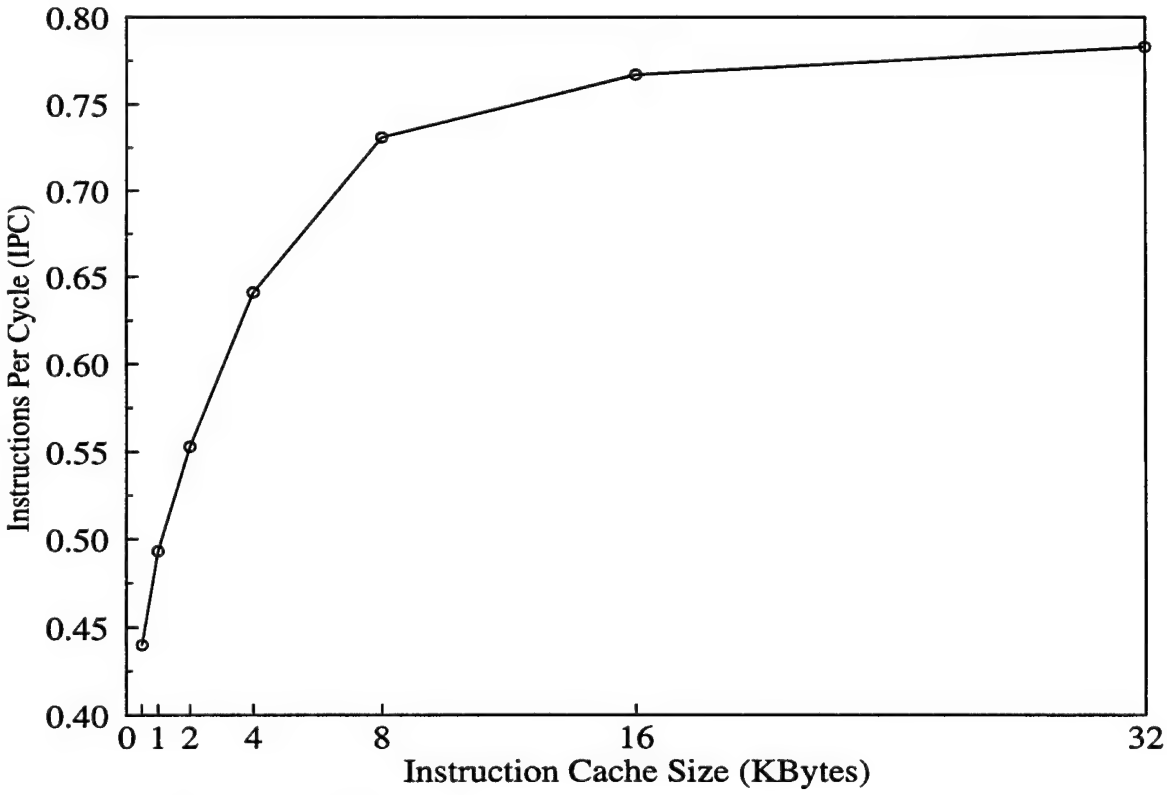


Figure 5.2: Effects of instruction cache size on overall performance.

Another important figure of merit for instruction cache performance is the instruction cache miss ratio, the ratio of cache misses to total number of accesses, plotted in Figure 5.3. While a 512 B instruction cache has only a 10% miss rate for the benchmarks used in this study, the corresponding performance, as shown in Figure 5.2, is poor. On the other hand an 8 KB instruction cache has a miss rate of 1.5%, offering a substantial performance improvement.

The results of Figures 5.2 and 5.3 demonstrate the trade-off between instruction cache size and performance. Designs with limited transistor budgets should incorporate a 4 KB instruction cache if possible, but need not increase the cache size beyond 8 KB. Increasing the cache size beyond 8 KB does not offer a substantial performance improvement.

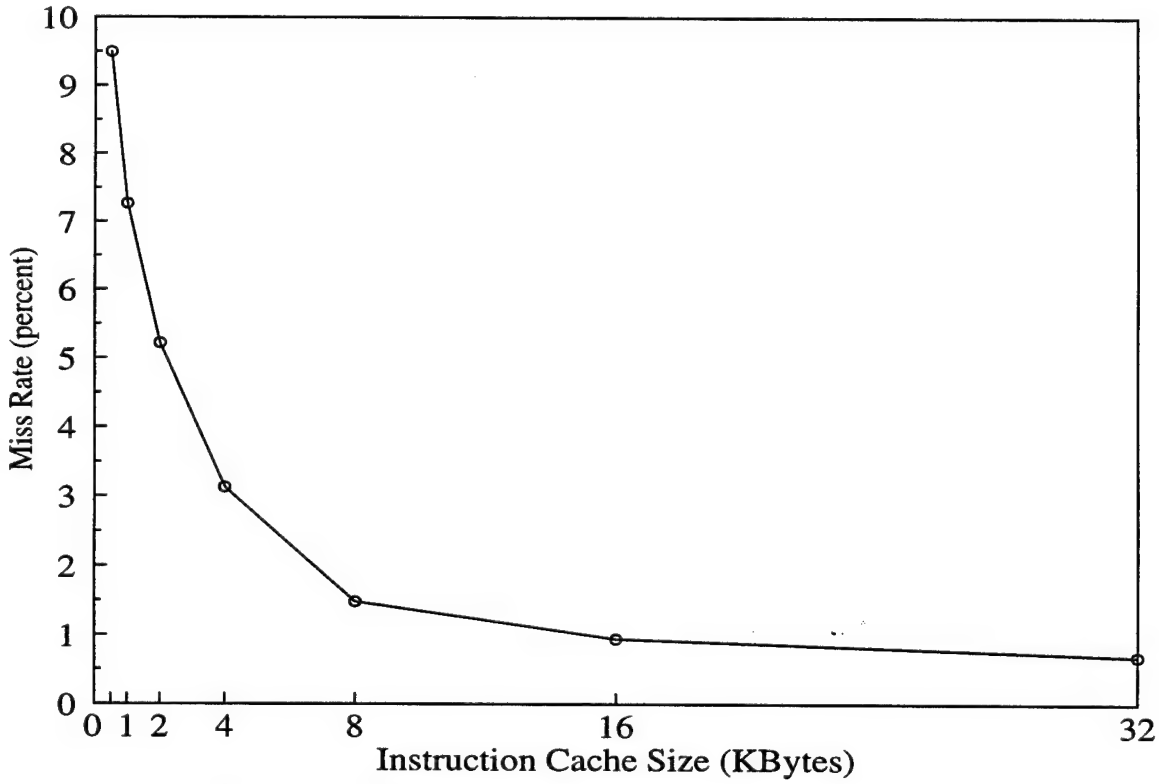


Figure 5.3: Effects of instruction cache size on miss rate.

5.3.1.2 Instruction Cache Line Size

Another important instruction cache parameter is line size. Figure 5.4 shows the simulated performance of the baseline processor as both the instruction cache size and line size are varied. The simulations reveal the optimal line size is 16 B, regardless of overall cache size. Line sizes that are either smaller or larger than 16 B degrade performance, although a larger line size is better than a smaller one.

Figure 5.5 plots the instruction cache miss rate for various instruction cache line sizes as the cache size is varied. A 16 B line size produces the smallest miss rate; however, the variation is less pronounced as the cache size is increased. Again, line sizes larger than 16 B are favored over smaller line sizes.

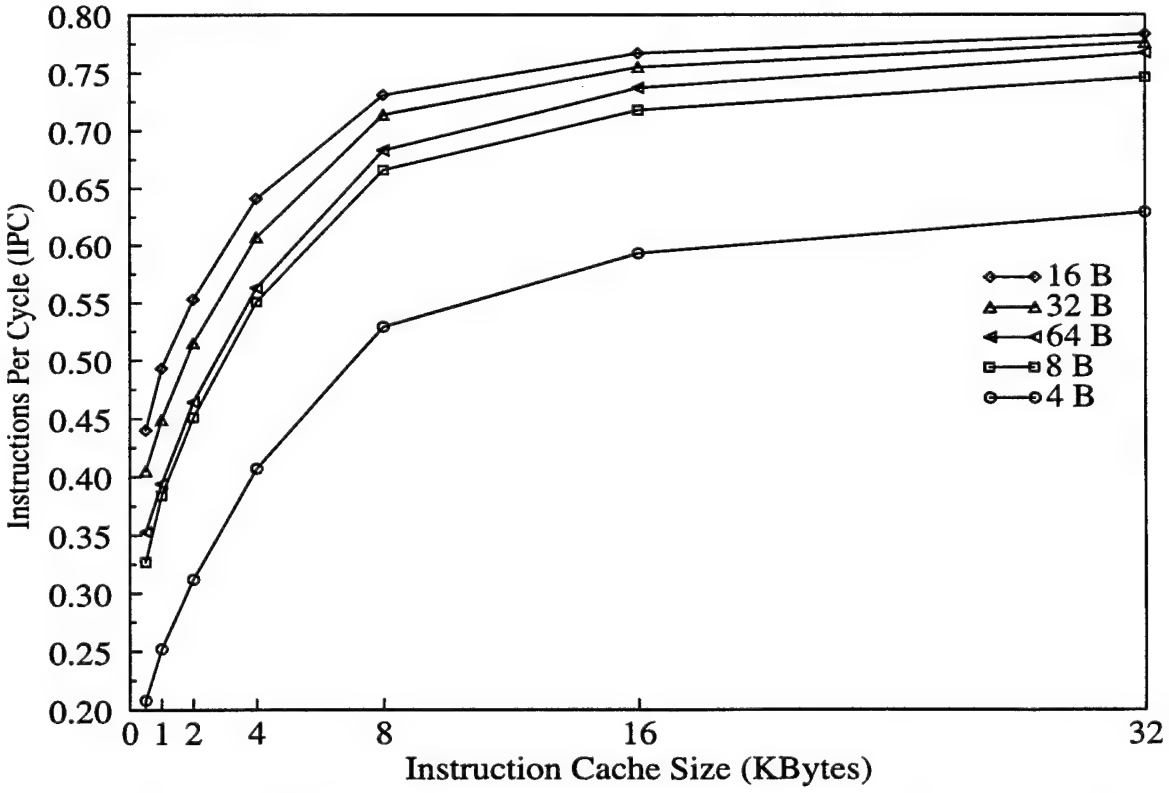


Figure 5.4: Effects of instruction cache line size on overall performance.

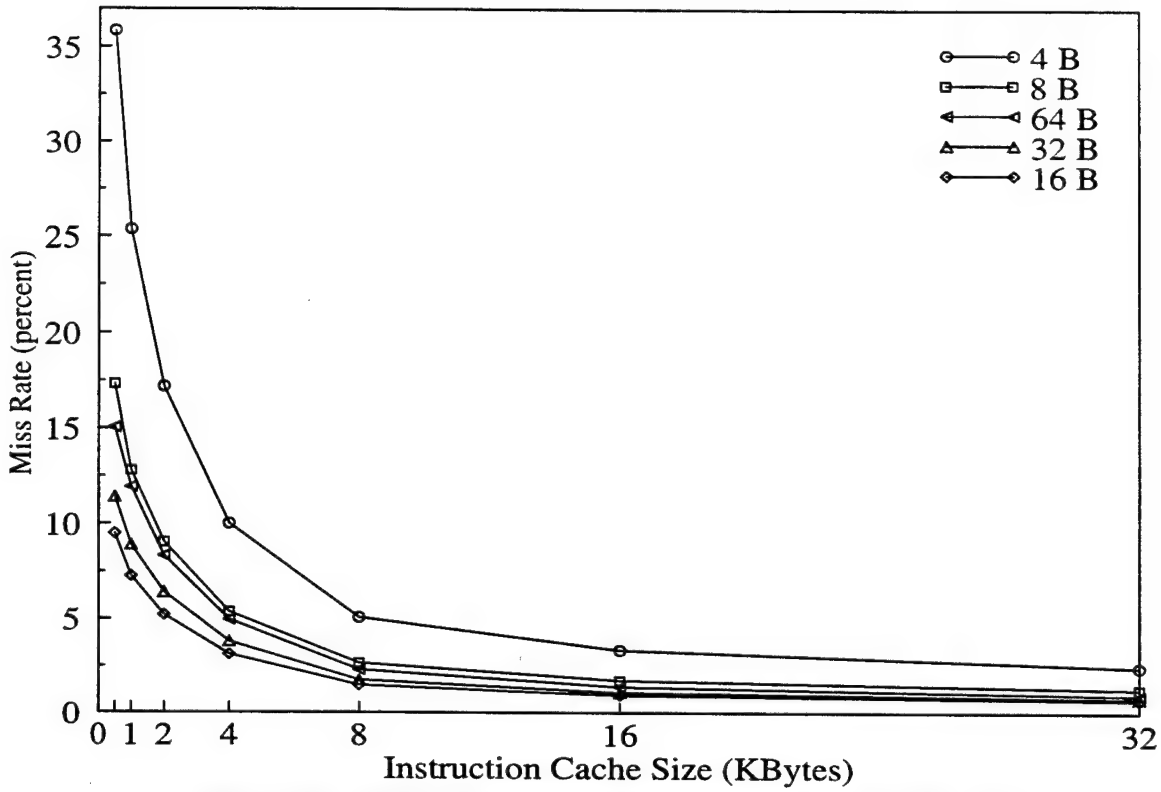


Figure 5.5: Effects of instruction cache line size on miss rate.

In both Figures 5.4 and 5.5 we observed that a 16 B line size offers optimal performance and miss rate. The reason for this has to do with the nature of most general purpose program applications. It is widely accepted that the basic block size of the average program is five instruction words. The simulations show that the optimal choice for the instruction cache line size is the equivalent of a basic block, about 4 or 5 instruction words (16 to 20 bytes). A line size that is too small results in an increased miss rate and reduced performance. On the other hand, a line size that is too large often results in unnecessary bus traffic, increased conflict misses within the cache, and internal fragmentation within the cache line.

5.3.1.3 Instruction Cache Associativity

Another trade-off of cache design is associativity. In general, a direct-mapped instruction cache has a shorter access time than a set-associative cache, but sacrifices performance. Figures 5.6 and 5.7 show the effects of instruction cache associativity on overall performance and instruction cache miss rates. In the simulations from which these figures were generated, the instruction cache associativity ranged between direct-mapped (1-way) and 8-way set associative, while the instruction cache size was increased from 512 B to 32 KB. The simulations reveal a noticeable difference in performance between a direct-mapped cache and a two-way set-associative cache. Higher levels of associativity offer little performance improvement over the two-way set-associative cache.

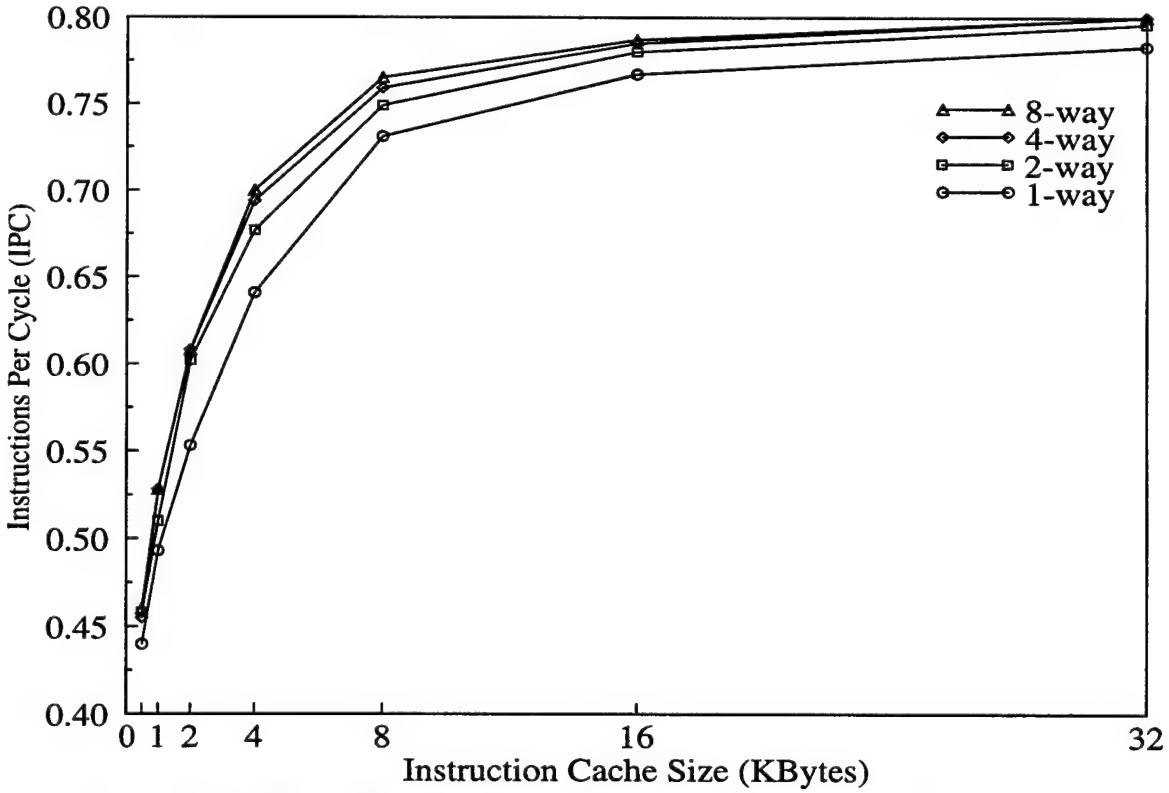


Figure 5.6: Effects of instruction cache associativity on overall performance.

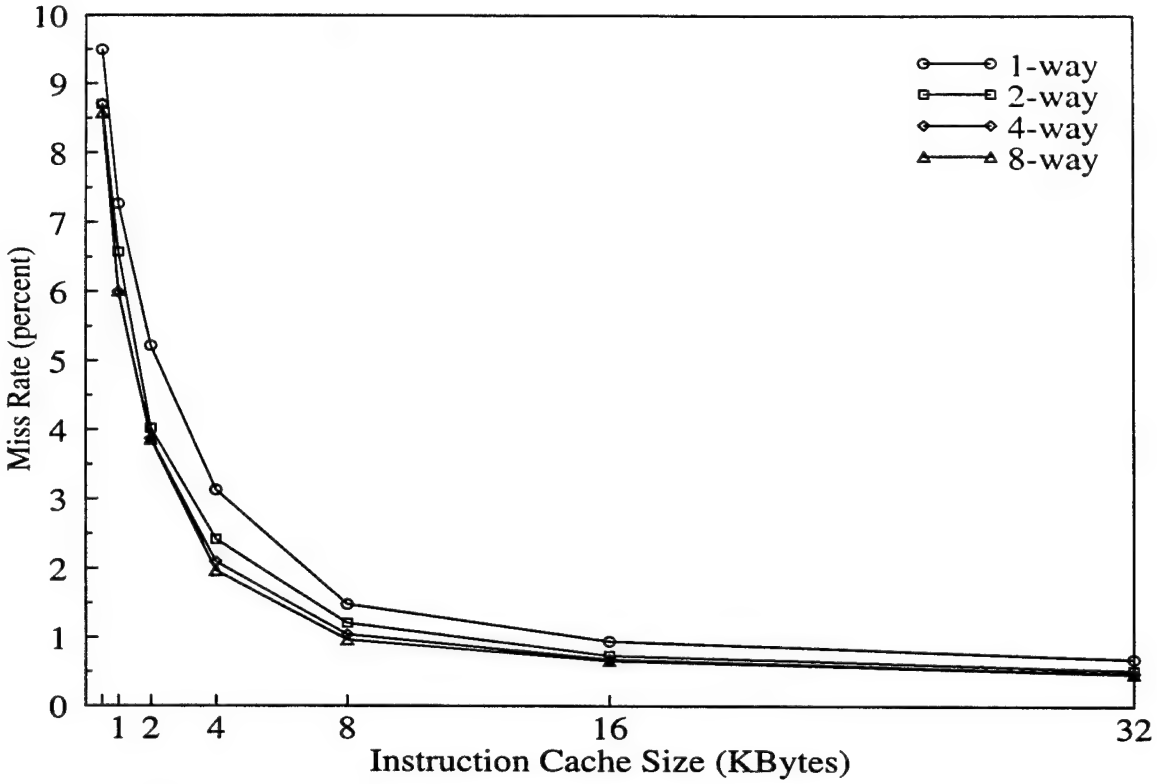


Figure 5.7: Effects of instruction cache associativity on miss rate.

For cache sizes 8 KB and larger there is a distinct performance advantage of increasing associativity rather than cache size. Increasing the associativity of large caches improves the performance as much, or more, than doubling the size of the cache; an 8 KB 8-way set associative cache is equivalent to a 16 KB direct-mapped cache. However, the same is not true of smaller caches. For instruction caches smaller than 8 KB increasing the associativity does not improve performance as much as doubling the size of the cache. While simulations show that small caches benefit more from increased size rather than increased associativity, cost sensitive applications may still benefit from increased associativity.

5.3.1.4 Instruction Prefetch with a Stream Buffer

The results of the previous studies have demonstrated how even a small instruction cache improves performance. As an alternative to simply increasing the size of the instruc-

tion cache, other means to improve performance were investigated. The performance of small on-chip instruction caches can be improved with additional software prefetch instructions [30], or hardware enhancements such as prefetch buffers [31]. Though a software-based approach would be advantageous for cost-sensitive applications, modifications to the instruction set would compromise compatibility with the PowerPC ISA. The stream buffer is a relatively inexpensive hardware modification, capable of providing significant performance advantages. A small stream buffer offers performance improvements similar to doubling the size of the primary instruction cache [31]. The stream buffer improves the effective miss rate and also hides some of the miss penalty associated with instruction cache misses. The stream buffer begins fetching consecutive lines of data from the location following a cache miss. When the fetch mechanism requests another data item that is not present in the cache, the stream buffer may be able to provide it. If the access hits in the stream buffer, then the line is supplied to the fetch engine and also copied into the instruction cache.

Figure 5.8 graphs the performance versus instruction cache size for a variety of stream buffer configurations. A single-entry stream buffer (1-entry) provides a large improvement over the baseline machine (0-entry). Stream buffers consisting of two and four entries provide additional performance improvements, but beyond four entries no noticeable performance gain is seen. The simulations show that, in general, a single-entry stream buffer can double the effective size of the instruction cache.

Figure 5.9 verifies that the stream buffer also improves the performance by hiding some of the latency incurred on an instruction cache miss. This figure shows processor

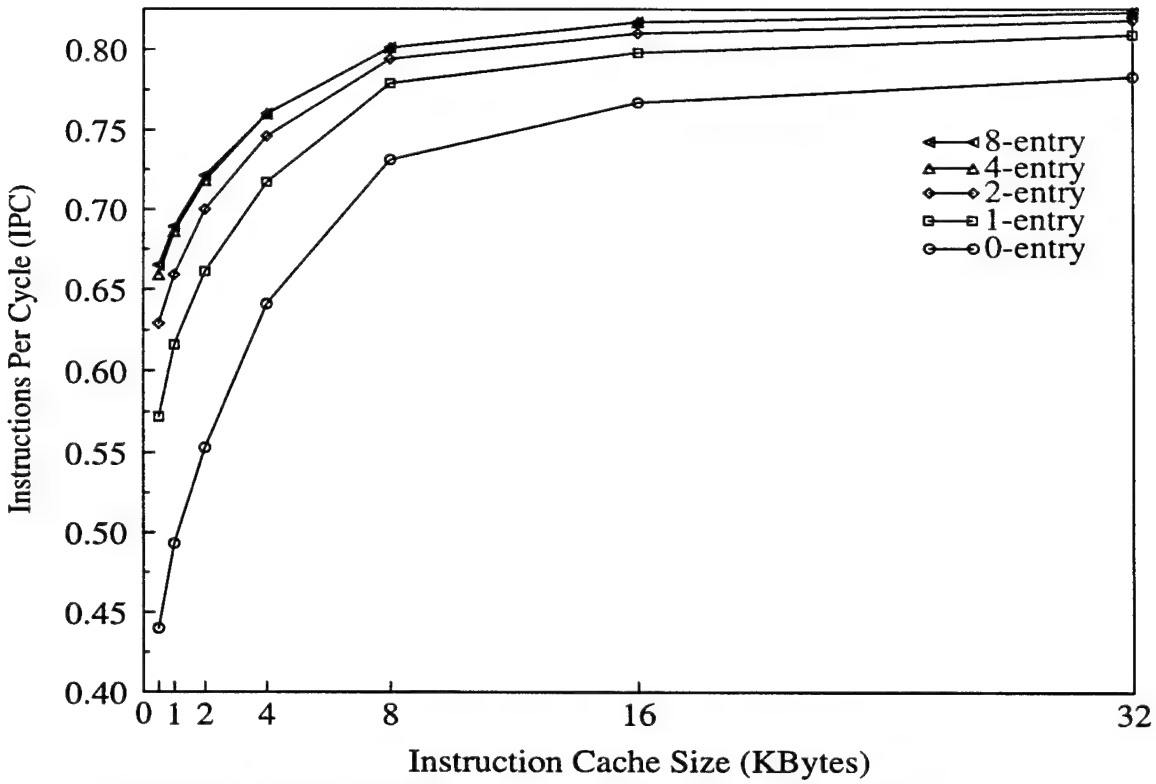


Figure 5.8: Stream buffer compensating for reduced cache size.

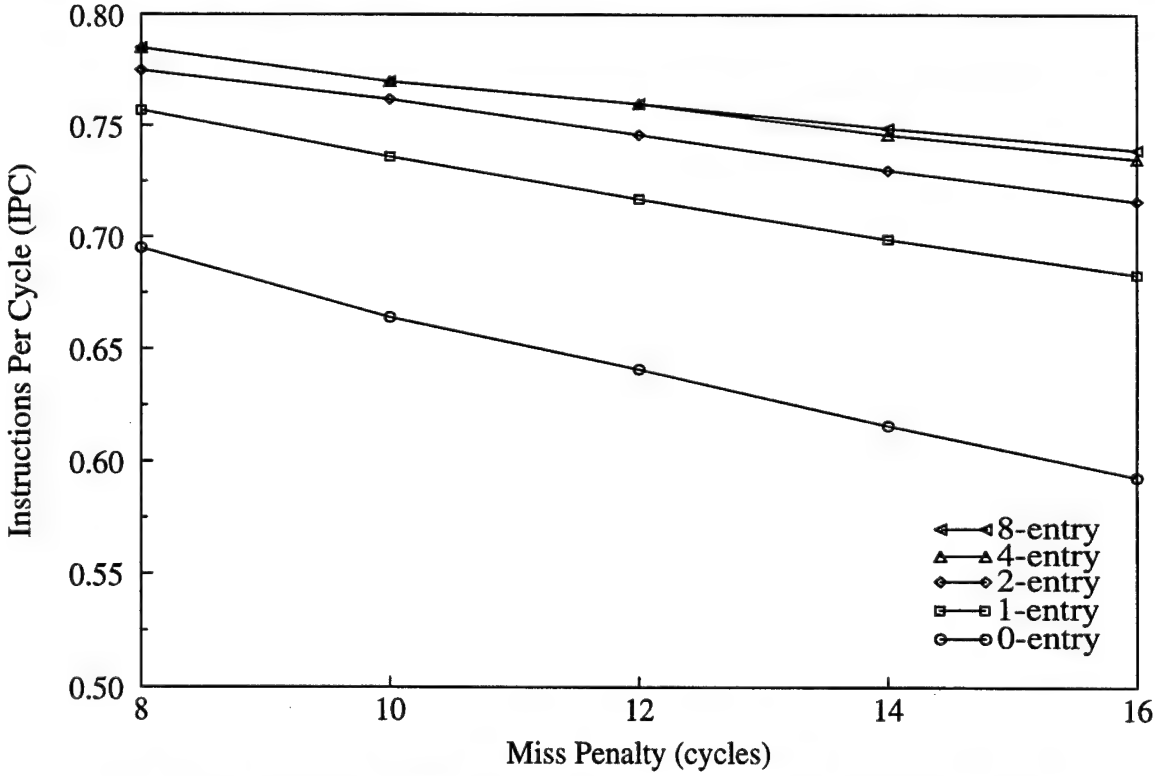


Figure 5.9: Stream buffer compensating for increased memory latency.

performance for various stream buffer configurations and instruction cache miss penalties. A single-entry stream buffer compensates for an extra eight cycles of secondary cache latency. Also note the slope of the performance curves corresponding to the various stream buffer configurations. The slope of the baseline processor (0-entry) is steeper than that of the other configurations, indicating that the stream buffer reduces the sensitivity of the processor to increased memory latency.

The stream buffer is a relatively inexpensive means of increasing instruction cache performance. The studies prove that a single-entry stream buffer more than doubles the effective instruction cache size, while also compensating for eight cycles of secondary cache latency.

5.3.2 Data Cache Configuration

The ability to load and store data is fundamental to processor performance. General purpose processors seldom have more than thirty-two registers, so most of the data must be stored in the primary data cache. The simple solution is to make the data cache as large as possible, but constraints of the CGaAs technology dictate that the primary data cache must either be small or else be located off-chip. An off-chip primary data cache increases the latency for memory accesses, leading to a potential performance penalty.

5.3.2.1 Data Cache Size

Unlike the instruction stream, data references do not always follow the principles of locality. The pattern of data references is highly dependent upon the application program. Therefore, it is much harder to consistently improve data stream processing. One guaran-

teed method of improving data cache performance is to increase the size of the data cache. The baseline machine configuration was simulated with data cache sizes ranging from 512 B to 32 KB. Figure 5.10 graphs the performance against primary data cache size. The graph demonstrates that increasing data cache size improves performance, at least to a cache size of 32 KB. Unlike the instruction cache, the data cache appears to have a higher upper-bound on cache size, beyond which further increases in cache size do not significantly improve performance.

Figure 5.11 graphs the data cache miss rate versus data cache size. The simulation results indicate that larger caches continue to substantially reduce the miss rate. Even a large data cache has a significant miss rate. A 32 KB data cache has a miss rate of almost 4.5%, leaving plenty of room for improvement.

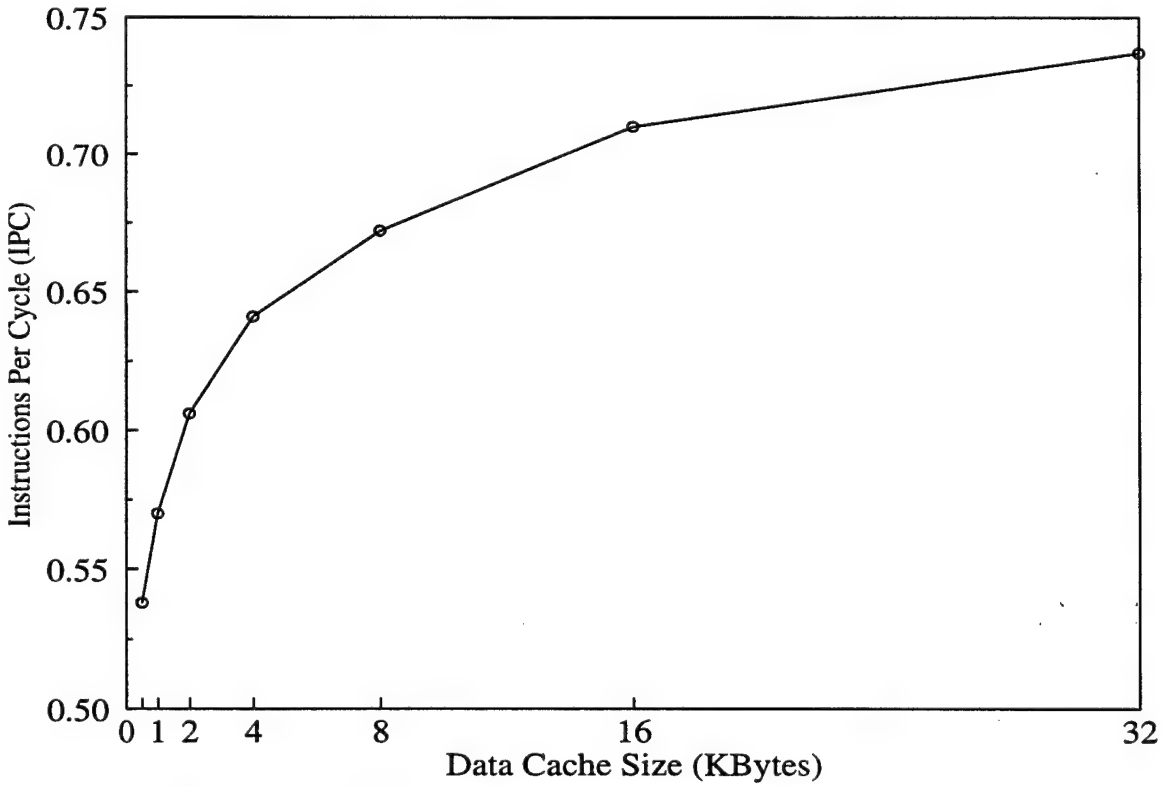


Figure 5.10: Effects of data cache size on overall performance.

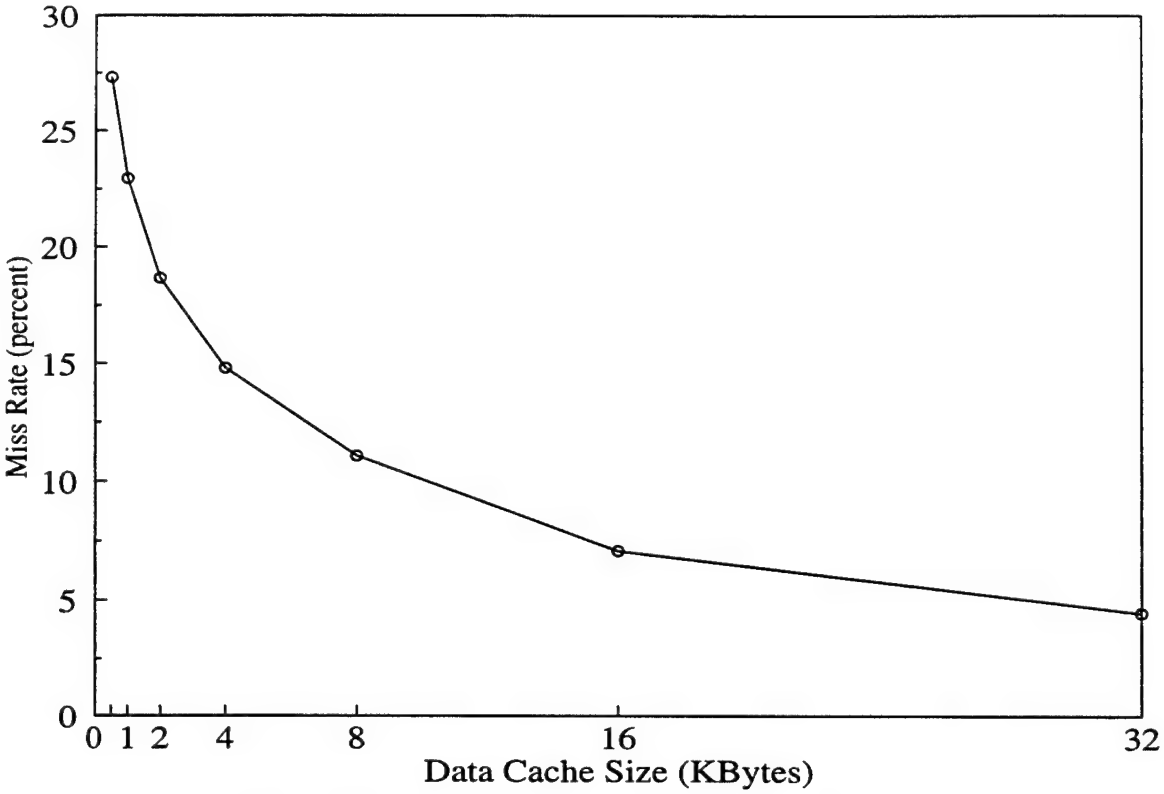


Figure 5.11: Effects of data cache size on miss rate.

5.3.2.2 Data Cache Line Size

It was noted earlier that the optimum instruction cache line size corresponded to a basic block. There is no fundamental reason that the data cache line size should have such an optimal size. Data access patterns are typically more varied and sparse than instruction accesses. The optimum data cache line size is highly dependent upon the specific application. The baseline machine configuration was simulated with a variety of data cache line sizes to identify the optimum line size for general purpose applications. Figures 5.12 and 5.13 reveal that line sizes less than 16 B can significantly reduce performance and increase the data cache miss rate. Line sizes of 32 B and 64 B offer an incremental improvement over a 16 B line size.

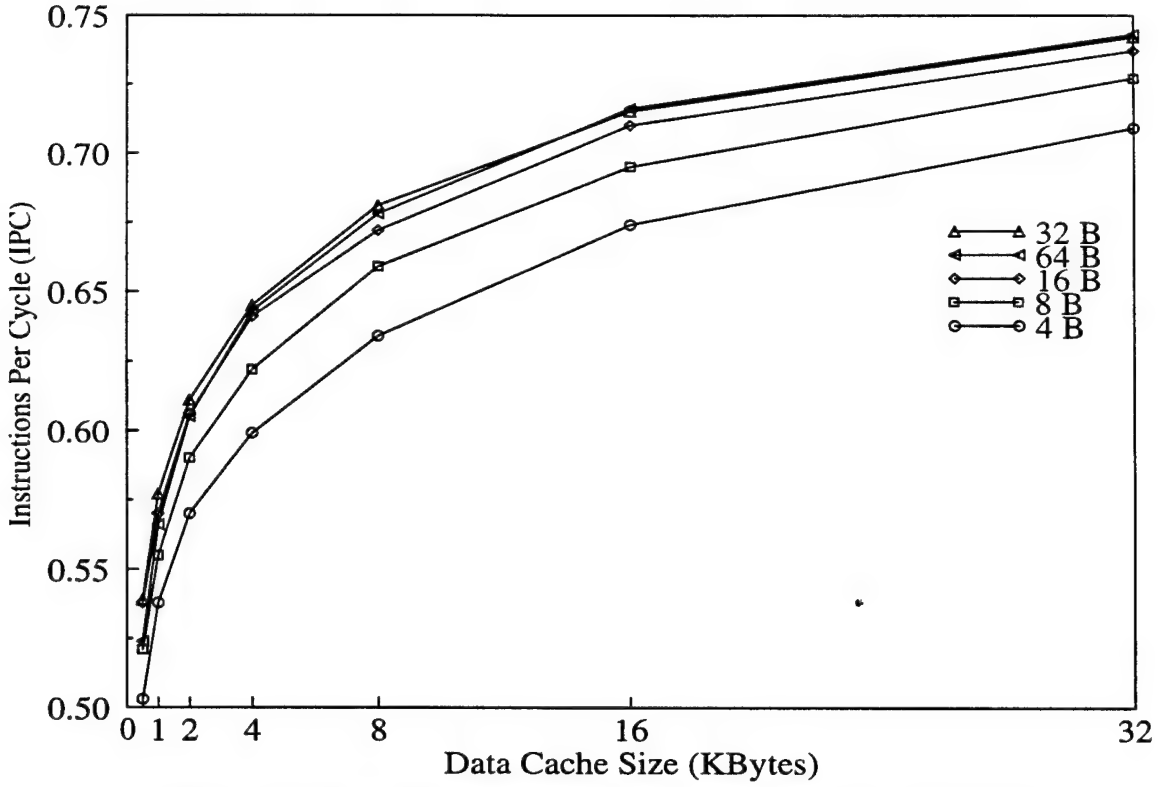


Figure 5.12: Effects of data cache line size on overall performance.

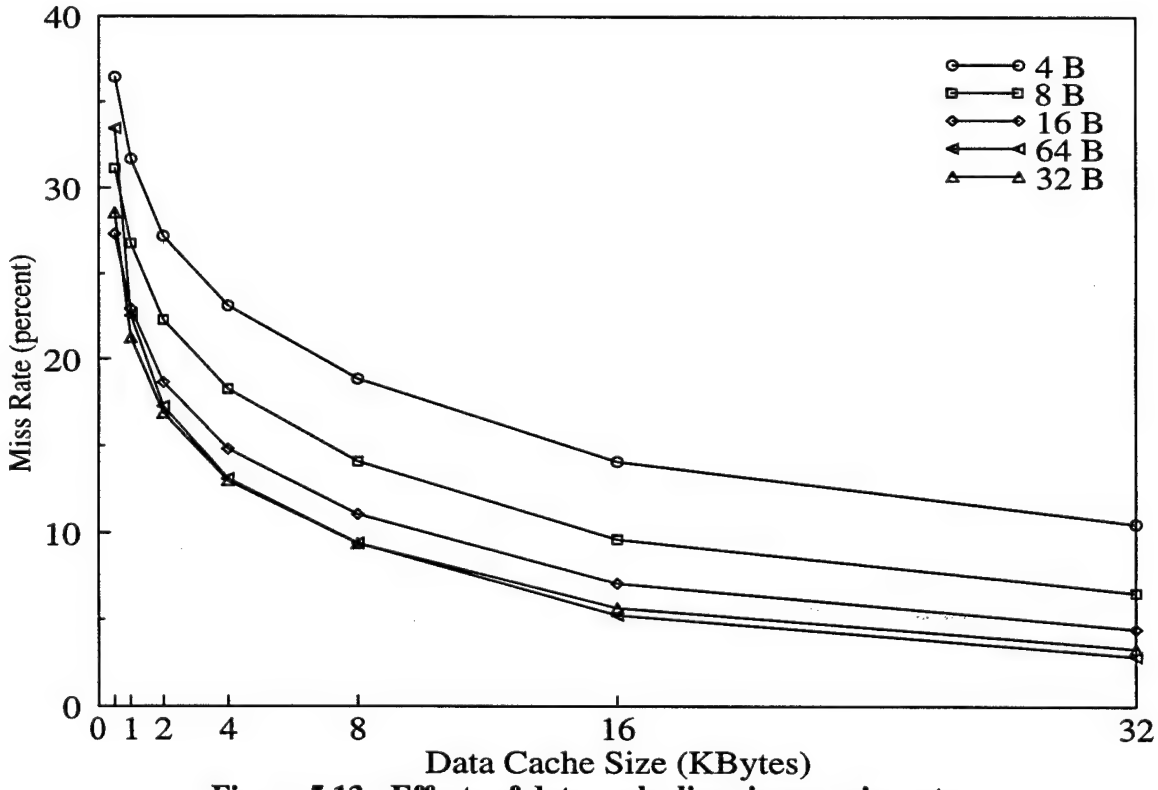


Figure 5.13: Effects of data cache line size on miss rate.

5.3.2.3 Data Cache Associativity

The baseline microprocessor was simulated with data cache associativity ranging from direct-mapped (1-way) to 8-way set-associative. Figures 5.14 and 5.15 plot the performance and data cache miss rates for various data cache sizes and associativities. The simulation results show a significant improvement from direct-mapped to two-way set-associative caches; higher degrees of associativity only marginally improve performance. In agreement with the familiar rule of thumb, a two-way set associative cache of a given size was found to offer the same performance as a direct-mapped cache that is twice as large [25].

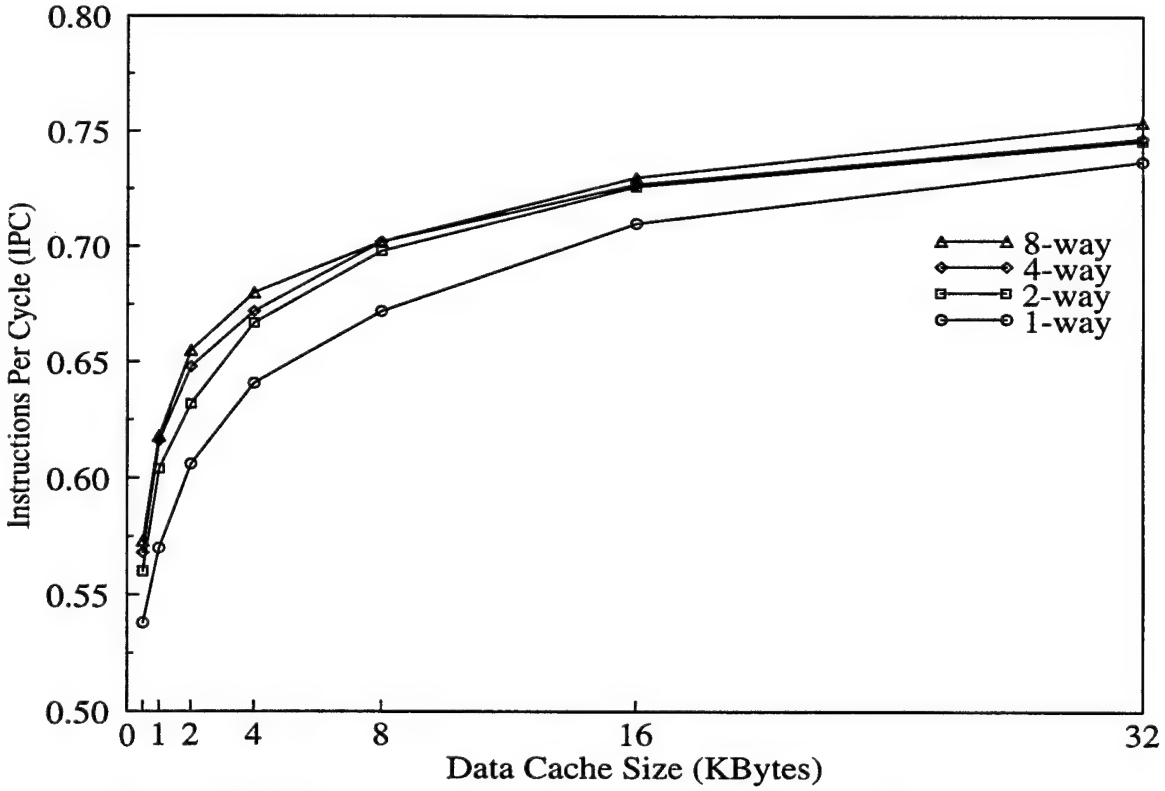


Figure 5.14: Effects of data cache associativity on overall performance.

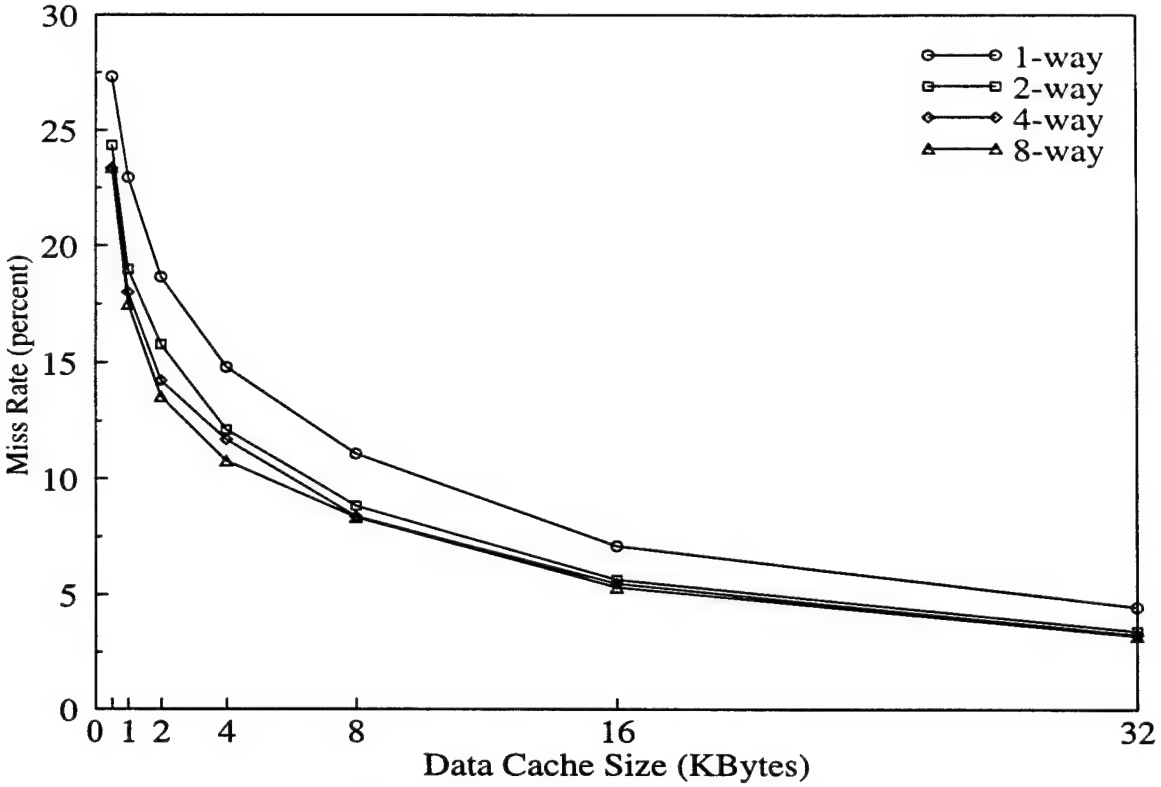


Figure 5.15: Effects of data cache associativity on miss rate.

5.3.2.4 Data Cache Latency

Section 5.3.2.1 demonstrated the importance of having a large data cache. The constraints of the CGaAs technology prohibit the implementation of a large on-chip data cache, forcing the primary data cache to be located off-chip. A large off-chip cache, by virtue of its size, can provide better performance than a smaller on-chip cache, provided the additional communication latency does not negate the performance advantage. Figure 5.16 plots the performance of the baseline machine with various data cache sizes and access latencies. An on-chip data cache configuration will have a single-cycle latency (1-cycle). An off-chip data cache may have a three-cycle latency (3-cycle), one cycle each for transmitting the index, accessing the array, and receiving the data. From these simulations, the general rule is that to maintain a given level of performance, the data cache must

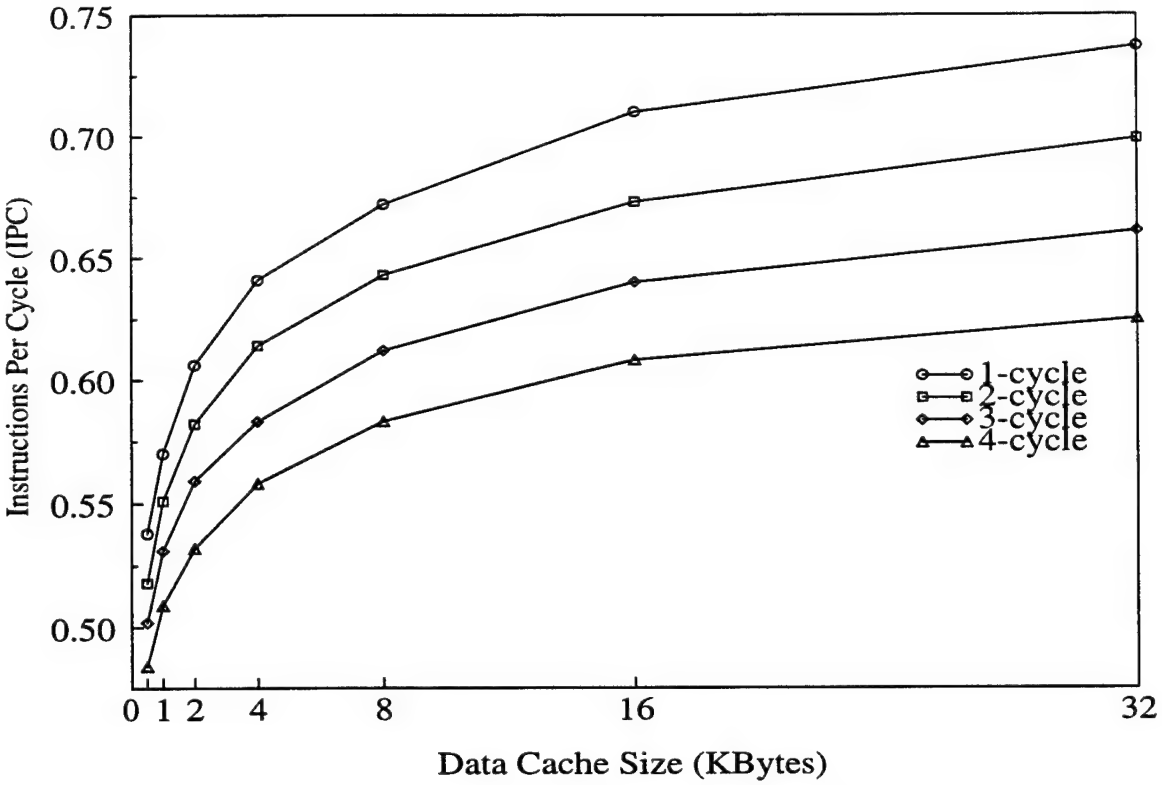


Figure 5.16: Effects of primary data cache latency on overall performance.

double in size for every additional cycle of latency incurred. An off-chip data cache must be 16 KB to provide the same performance as a 4 KB on-chip data cache. Microprocessors that are unable to incorporate a data cache on the main processor die can place the data cache off-chip provided that the interconnection is relatively high-speed and the cache is made large enough to compensate for the additional communication latency.

5.3.3 Optimizing Load and Store Operations

In addition to altering the data cache configuration, one can improve the overall performance by optimizing the processing of load and store requests. Store operations do not perform any useful computation, while load instructions are essential to the computation itself, providing the necessary operands. It follows that loads should be allowed to proceed before stores whenever possible. This optimization is commonly referred to as load

bypassing, that is, allowing loads to bypass stores whenever there is not an address conflict. A second improvement can be made in cases where there is an address conflict; the data from the store may be forwarded directly to the load. In addition to specific load and store enhancements, improvements can be made to all accesses in the presence of an outstanding cache miss. In a traditional pipeline, a data cache miss would cause the entire machine to interlock until the requested data is received. Using a technique referred to as miss scheduling, requests which hit in the cache are allowed to proceed while an outstanding miss is being serviced. Thus, the data cache interface is made non-blocking. This allows load operations to complete sooner, potentially allowing arithmetic computations to proceed while a cache miss is being serviced.

The baseline machine configuration was simulated with three types of enhancements to the load store unit; load bypassing, store forwarding, and miss scheduling. Figure 5.17 plots the overall performance improvement for each separate enhancement, while Figure 5.18 shows the relative improvement. The largest improvement is provided by load bypassing. Although the improvement is most evident for smaller caches which have more misses, load bypassing consistently improves performance by 2 to 3% for all cache sizes. Miss scheduling also provides up to 0.5% improvement for very small caches that have high miss rates, but as the cache size increases, the miss rate is reduced and the enhancement is not utilized as often. Store forwarding provides virtually no performance improvement. The findings are in agreement with those of Johnson [28], who also found that load bypassing is critical to performance, while store forwarding is not.

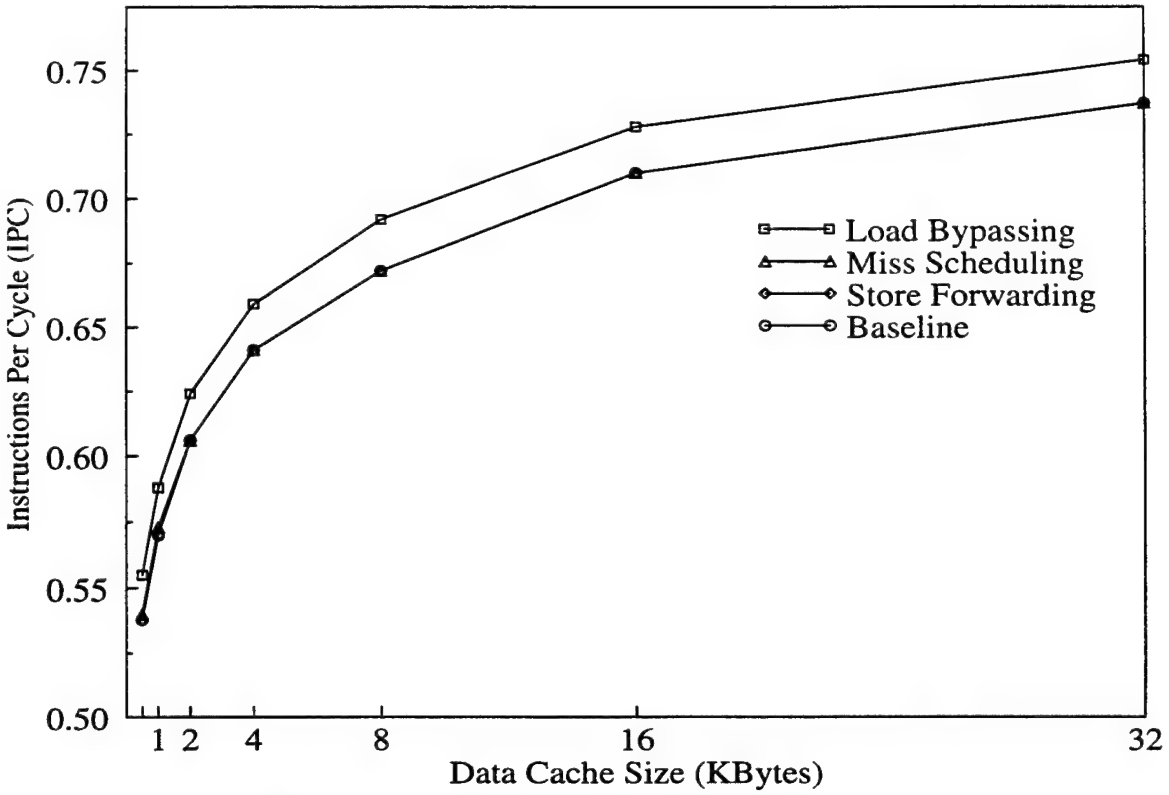


Figure 5.17: Effects of various functional unit enhancements on performance.

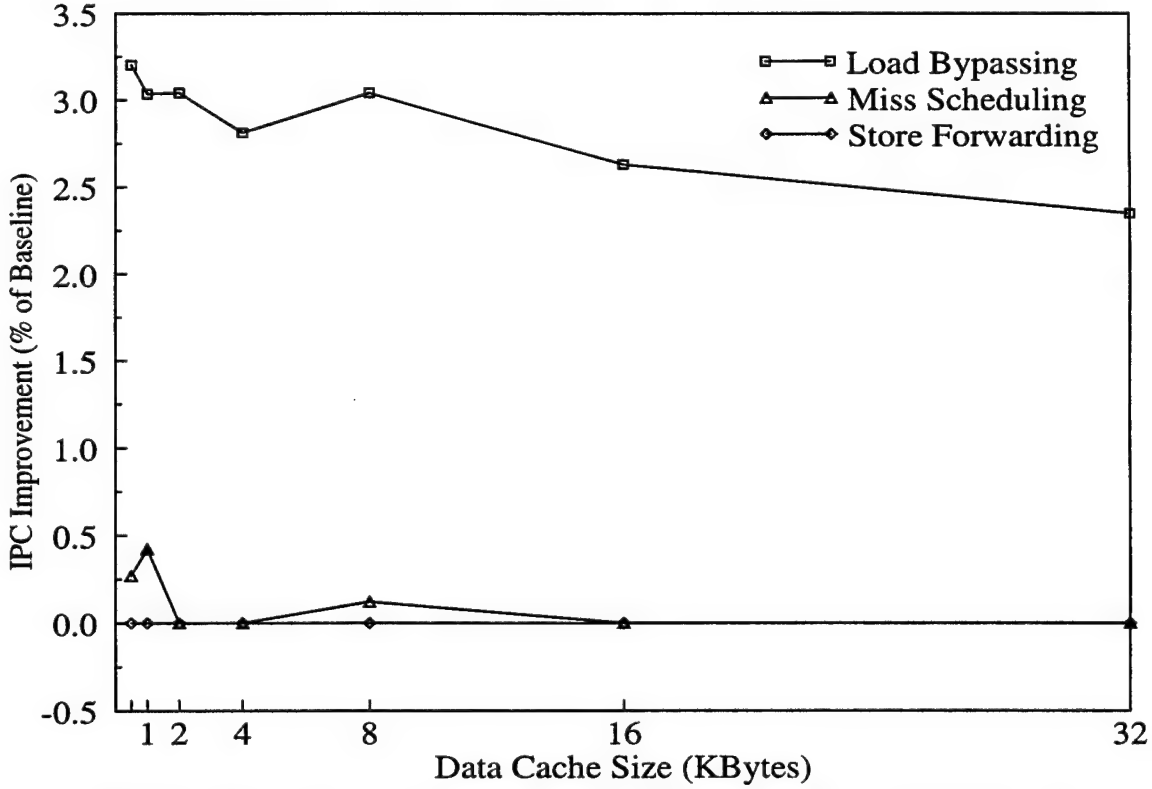


Figure 5.18: Improvement offered by various load-store enhancements.

5.3.4 Instruction Translation

Most RISC instructions perform one basic operation and modify a single architectural register. However, some PowerPC™ instruction variations perform multiple operations and modify several registers. The load and store instructions, for example, have forms that increment a base address register by the number of bytes accessed. Conditional branch instructions may decrement the count register during loop iterations and store the address of the next sequential instruction into the link register. Arithmetic instructions may be modified to compare the computed result against zero and set condition codes. The rotate instructions perform a complex sequence of operations that insert a rotated value from a source register into a destination register, under the control of a mask defined by the instruction word itself. While these compound instructions may be useful compiler primitives, they are not necessarily fundamental hardware operations. Performing compound operations in a single cycle may require additional hardware, additional cycle time, or both. For example, the designers of the Motorola 88000 cited hardware cost and complexity as a reason for not implicitly setting condition codes [32]. Designing an execution unit with a critical path derived from compound instructions results in wasted cycle time for simpler operations. On the other hand, extending the execution of specific instructions to multiple cycles complicates the stall mechanism. In addition, complex instructions often have multiple destination registers, complicating the dependency resolution and writeback logic.

Our decode mechanism solves these problems by translating complex PowerPC™ instructions into several simple unit-operations, each designed to require one minimal unit of computation time and modify only one architectural register. The minimal unit of com-

putational time is taken to be a simple arithmetic operation, such as a 32-bit add. This idea is implemented for a CISC instruction set in many of the high-performance x86 machines today. Unit-operations balance the critical path through the execution logic by decomposing compound instructions into fundamental operations. Dependency resolution is also simplified by the fact that each unit-operation produces only a single result. During the decode process a single PowerPC™ instruction may be translated to a sequence of unit-operations, the final unit-operation is designated by a flag in the decoded instruction word. The flag enables the resulting unit-operation sequence to be treated as an atomic operation; exception detection is suppressed until the flag signalling the PowerPC™ instruction boundary is reached.

Load and store with update instructions are decoded into a memory access and an add instruction. Branch instructions that decrement the count register are translated into a subtract instruction and the conditional branch. Arithmetic instructions with record are decomposed into a simple arithmetic operation and a compare instruction. The rotate instructions are translated into a sequence of instructions that perform the rotate, generate the mask, and insert the rotated value into the destination register under control of the mask. If necessary, the intermediate unit-operations may be designed to use a small set of physical registers for storing temporary results.

Most PowerPC™ instruction variations result in either a direct mapping to a single unit-operation or a translation to two unit-operations. Only the complex PowerPC™ rotate and mask instructions result in longer sequences of operations. Once a PowerPC™ instruction is translated into unit-operations the out-of-order superscalar core may extract

whatever parallelism exists within the architectural instruction itself. From our simulations, shown in Figure 5.19, the ratio of translated unit-operations to architectural PowerPC™ instructions was approximately 1.15 to 1. Although the translation from PowerPC instructions to unit-operations resulted in a 15% increase in the effective instruction count (the number of unit-operations), performance was only marginally impacted. The superscalar core is able to extract parallelism from the stream of unit-operations. This, coupled with the fact that compound PowerPC instructions are infrequent results in only a 2.8% degradation in overall performance, as measured by IPC.

5.3.5 Tuning the Superscalar Parameters

A number of different parameters affect the performance of superscalar microprocessors. The fundamental objective of the superscalar microprocessor is to exploit instruction

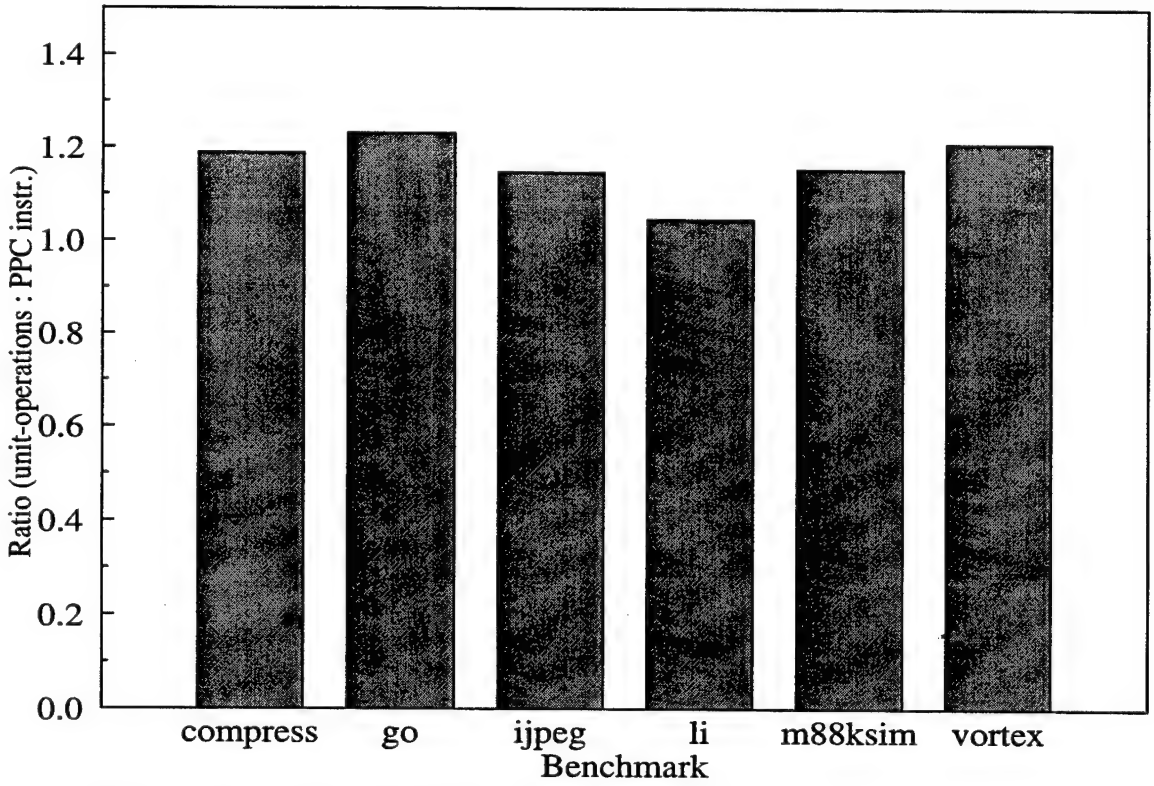


Figure 5.19: Ratio of unit-operations to PowerPC instructions.

level parallelism (ILP). Classical works [33,34,35] suggest that the average amount of ILP is approximately two, while recent microprocessor designs (with issue widths of four instructions per cycle and greater) are attempting to exploit the maximum, or peak, amount of ILP available. In practice, the amount of available ILP will vary depending upon the ISA, system configuration, and application program. In general, a machine must implement a large instruction window, and actively process many instructions simultaneously, in order to extract a significant amount of ILP and reach high levels of performance. Out-of-order execution is fundamental to superscalar performance, providing a speedup of as much as 2.5 over scalar machines [28]. Many methods for supporting out-of-order execution and precise interrupts have been developed. Dynamic scheduling mechanisms range from hardware intensive solutions such as checkpointing [36] and active lists [37], to simpler alternatives such as the future file, history buffer, reorder buffer [38,39], dispatch stack [40], register alias table and the register update unit [41].

By modifying the number of reorder buffer entries and reservation stations as well as the decode and completion width, the efficiency of the execution engine can be matched with that supported by the fetch mechanism. In the following studies we examine various aspects of the out-of-order superscalar execution engine. The baseline machine configuration uses a variant of the Tomasulo algorithm [42] in conjunction with a reorder buffer [38] to implement an out-of-order superscalar execution engine. The scheme utilizes reservation stations to buffer instructions at the input of each functional unit, and a reorder buffer to collect results from the functional units. The reorder buffer also provides a convenient means to implement precise interrupts.

The effects of superscalar machine width were studied by varying the width of the baseline machine; that is, the number of instructions the processor can decode, schedule, and complete per cycle. The width was varied from one to five. The sizes of the reservation stations, reorder buffer, and primary caches were made sufficiently large, so as not to obscure the effects of superscalar width. Perfect branch prediction was assumed, so as to supply the execution core with an uninterrupted stream of instructions. The results of the study are presented in Figure 5.20. Performance improves rapidly until the superscalar width reaches three, beyond that, little is gained unless complex instruction merging and aligning techniques are employed.

Perhaps as important as superscalar width is the amount of buffering present in the execution core. The reservation stations and reorder buffer allow the execution engine to

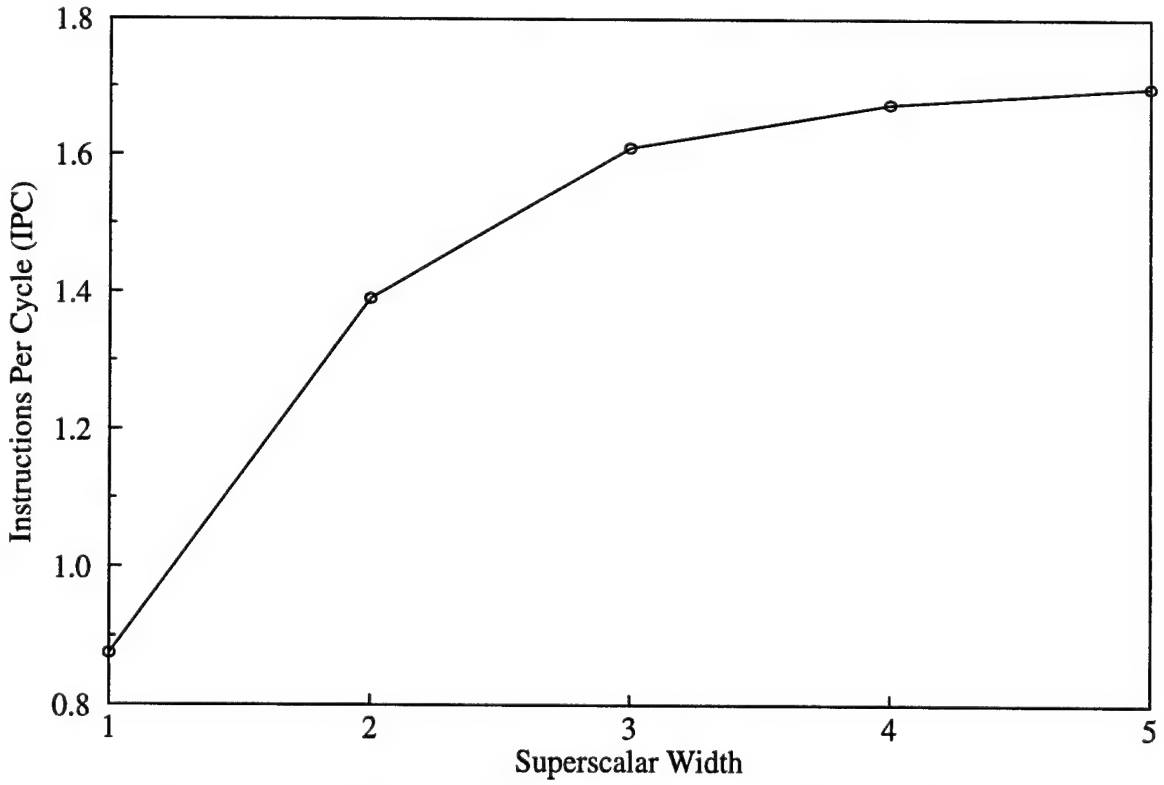


Figure 5.20: Effects of superscalar width on overall performance.

look ahead and extract parallelism from the instruction stream. If the buffering resources and the superscalar width are mismatched, the processor loses efficiency. In our model instruction issue is strictly in-order at the reservation stations to a particular functional unit. Allowing instructions to issue out-of-order at the reservation stations has been found to provide only a marginal increase in performance [28]. Figure 5.21 demonstrates the number of reservation stations per functional unit for varying degrees of superscalar width. For a one-wide out-of-order machine, two reservation station entries per functional unit suffices. For a two-wide machine, approximately three reservation stations are needed. There is little difference between the four and five-wide superscalar machines, both require about four reservation stations.

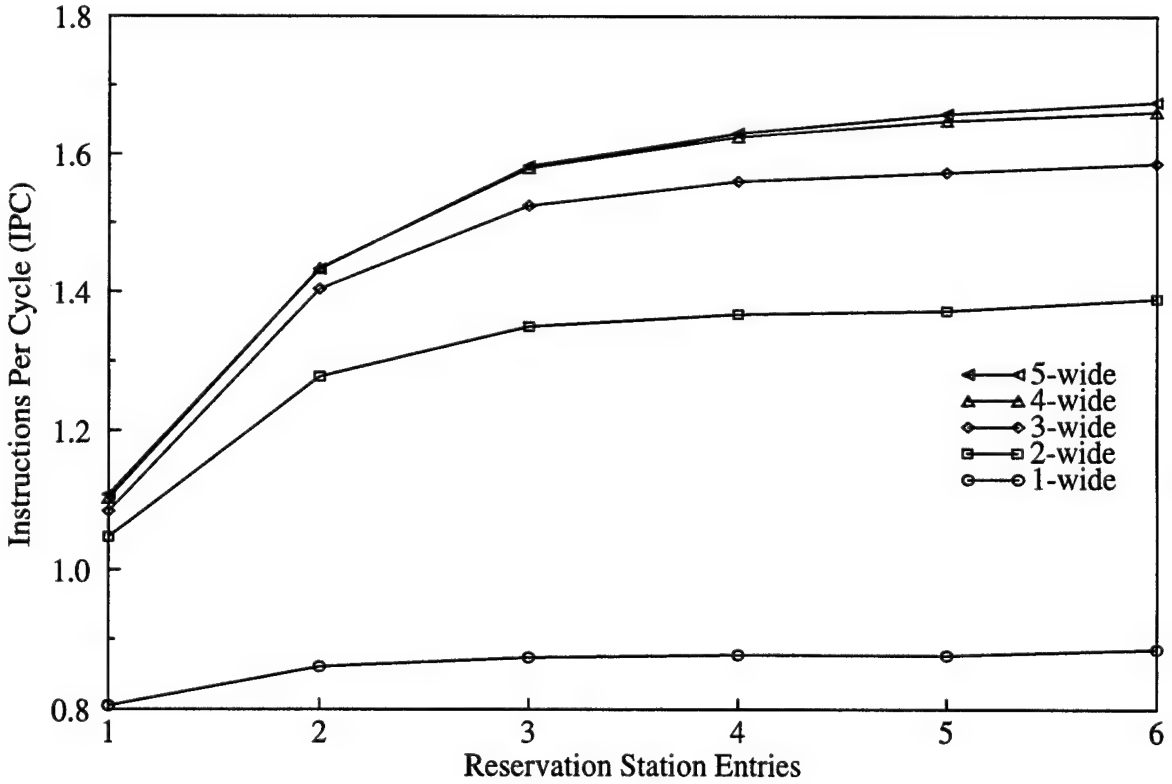


Figure 5.21: Effects of reservation station size on overall performance.

The function of the reorder buffer is to keep track of instructions in the active window. Figure 5.22 plots the reorder buffer requirements for varying degrees of superscalar width. A simple one-wide out-of-order machine requires only about eight reorder buffer entries to achieve the maximum level of performance, while the performance of a four or five-way superscalar machine continues to improve with more than 20 entries.

5.3.6 Dynamic Branch Prediction

Branch instructions represent control dependencies, and must be resolved before the correct path of instruction execution may be determined. The delay between instruction fetch and resolving the control dependency is referred to as the branch delay. This delay may result in several cycles of inactivity, or wasted execution slots, in a pipelined computer [43]. The cost of branches increases even more in superscalar machines. Superscalar

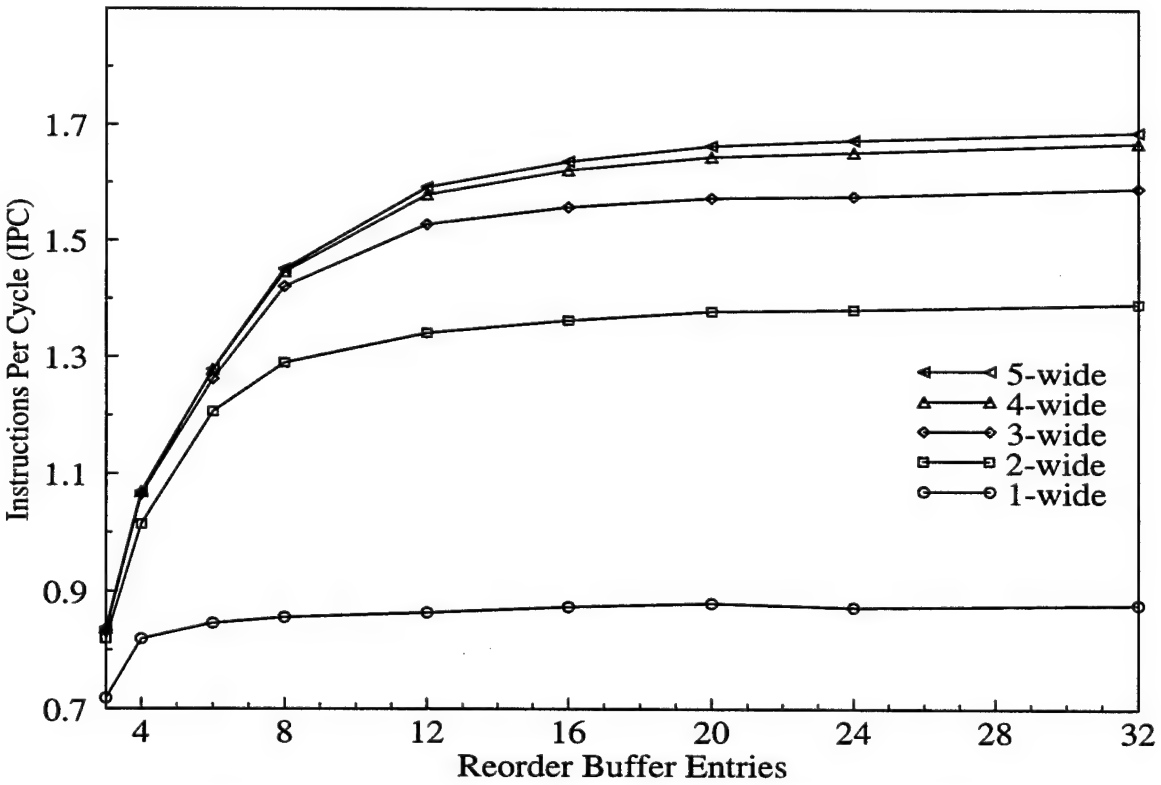
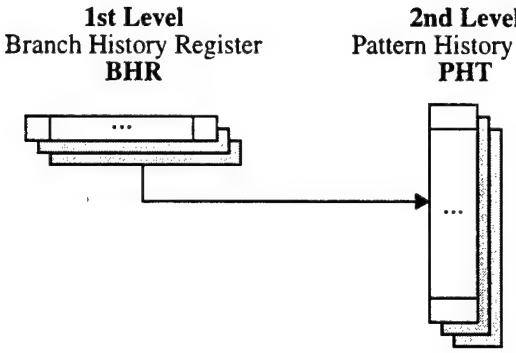


Figure 5.22: Effects of reorder buffer size on overall performance.

architectures are capable of dispatching several instructions per cycle, so an unresolved control dependency results in even more wasted execution bandwidth. The most common technique for improving the performance of a microprocessor in the presence of control dependencies is dynamic branch prediction.

Dynamic hardware branch prediction schemes rely upon the classic two-bit saturating counter [44]. The two-bit counter is used as a finite-state machine to encode the past behavior of a branch. The counter cycles through four states, ranging from strongly and weakly not taken to weakly and strongly taken. The idea is that two incorrect predictions are required to change the prediction of a counter that is in the strongly biased state. From this past behavior, future branch outcomes can be predicted.

Two-level adaptive branch prediction schemes [45,46,47,48], summarized in Figure 5.23, predict branches based on a pattern of recent branch outcomes. Information is stored in a two-level structure. The first-level consists of one or more branch history registers (BHR) that store the outcome of previous branch instructions. The second level is comprised of two-bit counters organized in a pattern history table (PHT) that dynamically predict a branch based on the pattern in a BHR. Two-level schemes differ in the method of storing first and second level information (i.e., the number and size of BHRs and PHTs). Global schemes store information across all branches and patterns. Per-set and per-address schemes use multiple BHRs and multiple PHTs to store information for either a set of branches, or a specific static branch. The original proposal of the two-level predictor suggested GAg, PAg, and PAp [45]. Two-level schemes can achieve a branch prediction accuracy of about 97%, but often require a significant amount of hardware to implement.



Variation	Description
GAg	Global branch history and global pattern history
GAs	Global branch history and per-set pattern history
GAp	Global branch history and per-address pattern history
SAg	Per-set branch history and global pattern history
SAs	Per-set branch history and per-set pattern history
SAp	Per-set branch history and per-address pattern history
PAg	Per-address branch history and global pattern history
PAs	Per-address branch history and per-set pattern history
Pap	Per-address branch history and per-address pattern history

Figure 5.23: Taxonomy of two-level adaptive branch predictor classification.

To accurately model the performance of the branch prediction schemes requires the simulation of tens of millions of branch instructions. The complexity of our superscalar simulator rendered the computation time of such a simulation prohibitively large. To facilitate the processing of a great many branch instructions, a custom branch-level simulator was written to process a dynamic stream of branch instructions and analyze the performance of various branch predictor configurations. The SPEC95 integer benchmark suite, summarized in Table 5.1, served as the benchmark for comparing the prediction accuracy of the schemes.

Benchmark	Static Branches	Dynamic Branches	Taken Branches	% Taken
compress	95	10,216,264	5,218,971	48.92
gcc	15,647	24,048,361	12,170,776	49.39
go	4,742	18,168,554	11,106,962	38.87
ijpeg	902	40,854,598	31,153,074	23.75
li	345	24,977,690	9,927,480	60.25
perl	1,576	31,309,305	15,507,818	50.47
vortex	5,963	24,979,201	7,302,625	70.77

Table 5.1: SPECINT95 branch characteristics.

While the branch problem can be solved with enough hardware, this becomes impractical in a small microprocessor design. Figure 5.24 shows the relationship between prediction accuracy (in terms of misprediction rate) and predictor cost. The configurations studied range from a simple one-dimensional array of two-bit counters [44] (Ag) to the

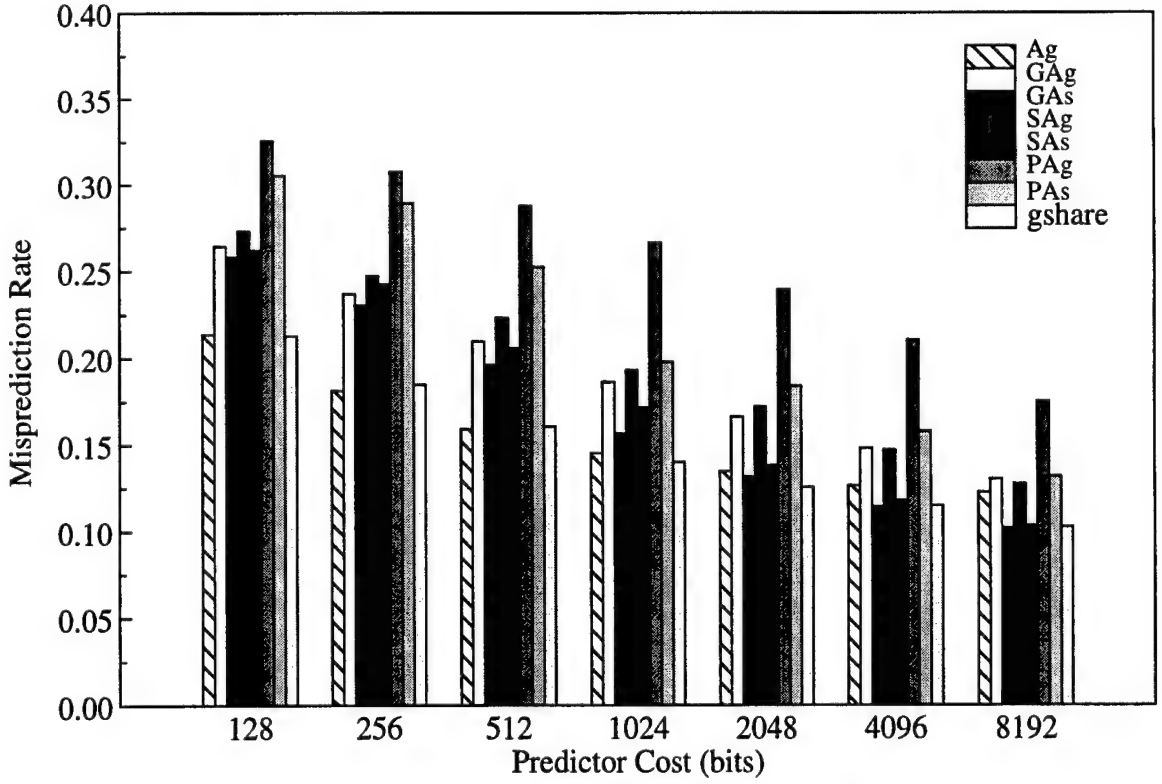


Figure 5.24: Effects of predictor cost on misprediction rate.

more complicated two-level adaptive schemes [45]. The one-level scheme outperforms all two-level predictors at very low costs (below 1K-bits). However, above 1 K-bits, the two-level schemes are generally superior. A 1 KB predictor implementing global sharing (gshare) [49] can achieve about 90% accuracy; this is a simple and efficient enhancement to a small microprocessor design. The gshare predictor is basically an implementation of the GAg scheme where the PHT is indexed using a hash of the program counter and the BHR. GAs and SAs also achieve approximately 90% accuracy as well, but they require a multiple PHTs, complicating implementation.

Figure 5.25 plots the overall performance for varying branch prediction accuracies and superscalar widths. An infinite buffer size model was assumed, as in Figure 5.20. The simulator resolves branches during the execute stage, creating a three-cycle penalty for

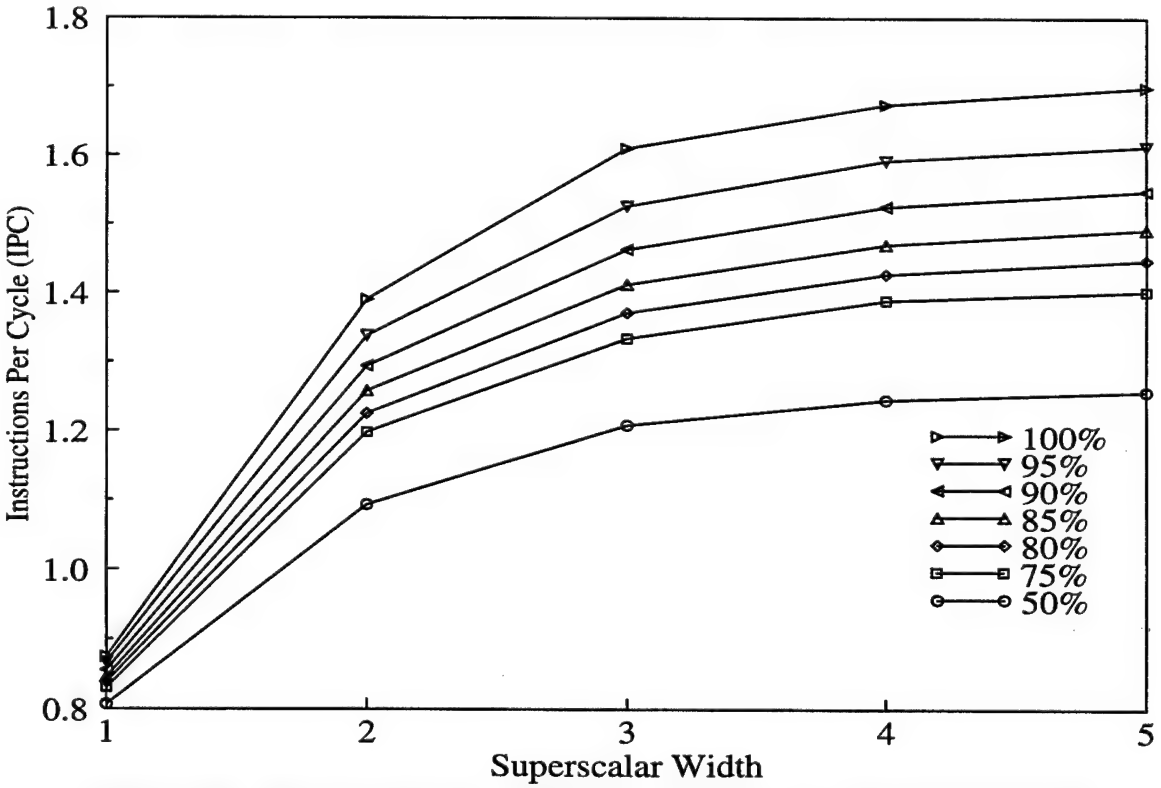


Figure 5.25: Effects of branch prediction accuracy on overall performance.

mispredicted branches. As the degree of superscalar width increases, branch prediction accuracy becomes more crucial to performance. Increased branch prediction accuracy does little to improve performance in a one-wide machine, while higher-order superscalar machines show significant performance improvements. The asymptotic IPC for perfect branch prediction (100%) is approximately 1.7, indicating an upper bound on issue width, beyond which high-degree superscalar machines will not provide further performance improvements. This also demonstrates the limited scalability of the benchmarks and the inability of the compiler technology to enable an execution rate of more than 2 IPC, even in an infinitely wide superscalar design. It should also be noted that accuracies of 95% to 100% are very difficult to achieve and require significant amounts of hardware, while accuracies of 90% are readily attainable, as Figure 5.24 demonstrated. The trends in Figures 5.24 and 5.25 indicate that a relatively inexpensive 1 KB branch predictor, achieving 90% accuracy, can double the performance of a two-wide superscalar microprocessor.

5.3.7 The Overall Effects

The previous experiments explored the performance improvements afforded to the baseline microprocessor by various architectural features and enhancements. The results of the individual studies can be combined to create a complete microarchitecture specification. Here we construct several microprocessor configurations and compare the relative performance and cost. Five machine configurations were constructed, ranging from extra-small (XS) to extra-large (XL) (500,000 to 10,000,000 transistors).

The various processor models are summarized in Table 5.2. The estimated transistor counts were arrived at based upon the number of the various on-chip resources required

and similar design examples. While we are specifically concerned with microprocessors requiring no more than one million transistors, we have included larger models for the sake of comparison. The extra-small processor does not have an on-chip data cache; instead it utilizes a 16 KB off-chip (3-cycle latency) cache. All cache structures are 2-way set associative. The large and extra-large processor models contain an extra integer unit, align instructions and merge correctly predicted instruction runs prior to decode, allowing them to fully utilize the available superscalar width. The baseline microprocessor configuration proposed in Section 5.1 is included as a reference point. Table 5.2 also rates the performance efficiency of the various microprocessor configurations. The efficiency is given in IPC per one-million transistors. This metric gives some insight into how the microprocessor achieves higher levels of performance. The extra-small and small microprocessor models are fairly efficient compared to the baseline microprocessor. The medium, large, and extra-large configurations are all less efficient than the baseline model, and considerably less efficient than the extra-small and small configurations. However, for general-purpose desktop microprocessors this inefficiency can be tolerated because of the high performance delivered. In cost-sensitive applications the efficiency is critical.

Parameter	Baseline	XS (extra-small)	S (small)	M (medium)	L (large)	XL (extra-large)
Superscalar width	2	2	2	4	5	6
Branch predictor (KB)	0	0.256	1	1	2	2
Branch predictor accuracy (% correct)	na	87	90	90	92	92
Reorder buffer (entries)	16	8	12	20	32	32
Reservation stations (per functional unit)	4	2	3	4	4	4
Branch target buffer (entries)	0	64	128	256	256	256
Dcache size (KB)	4	0	4	16	32	64
Stream buffer (entries)	0	2	4	4	4	4
Icache size (KB)	4	1	2	8	16	64
Transistors (K)	1035	503	1003	2459	4355	9989
Instructions Per Cycle (IPC)	0.6410	0.7642	0.9296	1.4575	1.7356	1.8447
Efficiency (IPC/M-transistors)	0.6193	1.5193	0.9268	0.5927	0.3985	0.1847
MIPS (at 200 MHz)	128	153	186	292	347	369
Advantage over sequential pipeline	1.1033	1.2749	1.3996	1.8082	2.0356	2.0849
Advantage over Baseline configuration	1.0000	1.1922	1.4502	2.2740	2.7076	2.8778

Table 5.2: Performance of various microprocessor configurations.

The performance of the various machine configurations is plotted graphically in Figure 5.26. The configurations are represented by the bars labeled PowerPC. The performance of the baseline microprocessor configuration is indicated by a dashed line. The results underscore the importance of instruction prefetching and dynamic branch prediction, even for small designs. The extra-small and small processors incorporate both of these features, albeit on a small-scale, and outperform the baseline microarchitecture despite the inferior instruction cache size. The medium, large, and extra-large processors

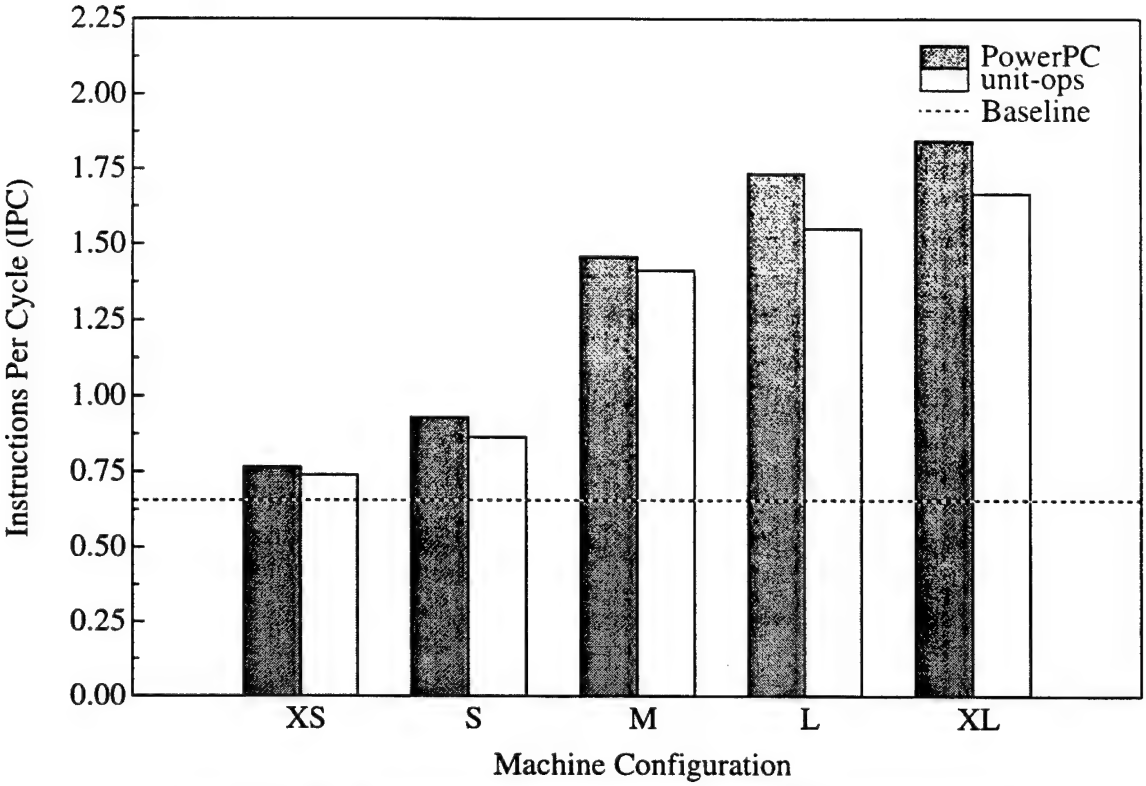


Figure 5.26: Performance of various machine configurations

significantly outperform the extra-small and small machines, but at a significantly higher hardware expense. For example, the extra-large processor outperforms the small processor by a factor of two, but requires ten times the number of transistors to implement. From our design perspective, the extra transistors are not being utilized efficiently and therefore the increased cost cannot be justified. Also notice that the extra-large processor is well past the point of diminishing returns, as it uses twice as many transistors as the large processor, yet only improves performance by 6%.

Also shown in Figure 5.26 is the performance of the machine configurations when the unit-operation mechanism (unit-ops) described in Section 5.3.4 is incorporated into the design. The penalty for translating PowerPC instructions to simpler unit-operations is only 2.8% of the overall performance. This feature simplifies the design of the dependency res-

olution, writeback, and execution logic, thereby increasing the clock frequency of the design. While the translation reduces IPC, it also provides a potential reduction in cycle time. The end result is likely a speedup in execution time for a given application.

The five machine configurations were also compared against a comparable sequential pipelined microprocessor. In each case the sequential pipelined processor was configured with instruction and data cache sizes and instruction latencies identical to the superscalar counterpart. The results are summarized in Table 5.2 and Figure 5.27. All five machine configurations provide significant improvement over a sequential pipeline. Improvements range from a 30% speedup for the extra-small processor to a 109% speedup for the extra-large processor configurations.

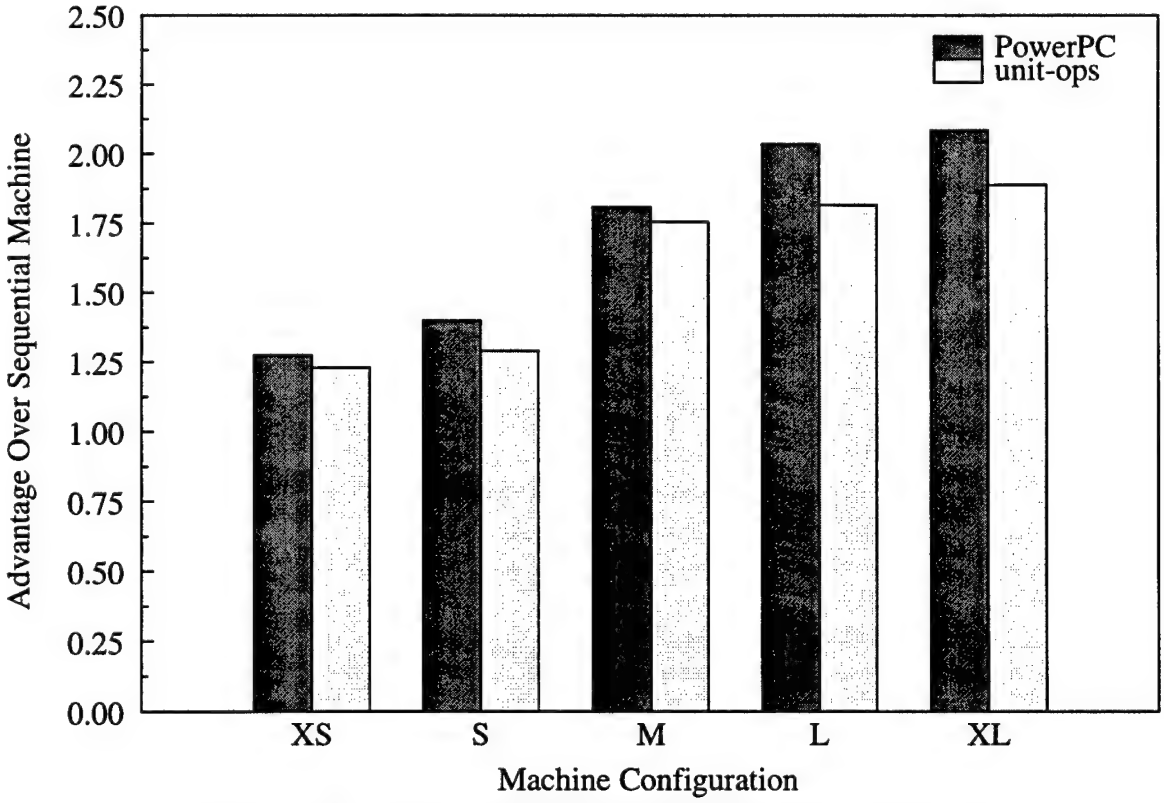


Figure 5.27: Advantage over sequential pipelined machine.

5.3.8 The CGaAs PowerPC Microprocessor Architecture

The simulations indicate that both the extra-small and small microprocessor configurations described in Section 5.3.7 are efficient superscalar implementations, from a performance per transistor standpoint (IPC/million-transistors). At this point in the development of the CGaAs PowerPC Microprocessor it was apparent that the CGaAs technology would not be able to support a one-million transistor IC. The extra-small microprocessor configuration was settled upon as the proposed CGaAs PowerPC Microprocessor architecture because it required only 500,000 transistors. The proposed microarchitecture is characterized by a small on-chip cache, a small two-level dynamic branch predictor, an off-chip data cache, and a simple dual-issue superscalar execution scheme. The 1 KB on-chip instruction cache is backed by a two-entry stream buffer. The instruction fetch mechanism is guided by a 256-Byte two-level dynamic branch predictor implementing global sharing, capable of predicting branches with 87% accuracy. The 16 KB primary data cache is located off-chip, presenting a three-cycle memory access latency. The primary data cache latency is overcome by a pipelined load-store unit, load and store enhancements, and the superscalar nature of the execution core. The dual-issue out-of-order superscalar execution scheme requires only an eight-entry reorder buffer and two reservation stations per functional unit. The proposed architecture is expected to achieve an execution rate of 0.76 IPC, representing a 27% improvement over a comparable sequential pipelined microprocessor, and a 20% improvement over the baseline microprocessor configuration described in Section 5.1. The proposed architecture, running at 200 MHz, will achieve an estimated 153 MIPS.

5.4 Summary

The goal of this chapter was to optimize the microarchitecture of a sub-million transistor superscalar microprocessor. A dual-issue superscalar microprocessor with dual 4 KB instruction and data caches was used as the baseline microprocessor configuration. Various features and enhancements were added to the baseline model, and the resulting performance was analyzed.

The performance of small primary instruction caches was investigated. Simulations found that performance could be improved dramatically by instruction prefetching. Simulations of the architecture demonstrated that a single-entry stream buffer provided a performance improvement equivalent to doubling the size of the instruction cache.

The small transistor budget prohibited the implementation of an on-chip primary data cache. Communication between the processor and the off-chip primary data cache will require a multi-cycle memory access. Simulations of the configuration revealed that for every additional cycle of latency incurred, the cache must double in size to maintain performance. Improvements in the Load-Store Unit were also investigated as a means to further improve data cache performance. Load bypassing and miss scheduling were two features found to improve the performance of load and store operations. Store-forwarding was found to not significantly improve performance.

The concept of unit-operations was introduced to balance and simplify the implementation of the PowerPC ISA. Translating compound PowerPC instructions into unit-operations results in a sequence of fundamental RISC operations which balance the cycle time consumed during computation, while writing only a single architectural register. By elimi-

nating the need for additional register file ports and the long critical paths associated with compound instructions, the unit-operation approach increases clock frequency over what it would otherwise be. The translation from PowerPC instruction to unit-operations results in a 15% increase in the effective instruction count. However, the superscalar core is able to extract parallelism from the stream of unit-operations. This, coupled with the fact that compound PowerPC instructions are infrequent results in only a 2.8% degradation in IPC.

The fundamentals of superscalar execution were investigated. Simulations analyzed the performance of various degrees of superscalar execution and buffering resources. It was found that a small microprocessor could support a dual-issue out-of-order superscalar execution core efficiently with only eight physical registers and two reservation stations per functional unit.

It was demonstrated that dynamic branch prediction can be efficiently implemented with few transistors. Simulations evaluated a variety of one- and two-level branch prediction schemes. It was found that a 1 KB two-level dynamic branch predictor implementing a global sharing policy could achieve 90% prediction accuracy, while a 256 B structure could attain 87% accuracy.

These architectural enhancements were combined to create several microprocessor configurations with implementation costs ranging from 500,000 to ten-million transistors. Architectures utilizing these enhancements and requiring less than one-million transistors were found to have high degrees of efficiency, as measured by a performance per transistor metric. A microarchitecture for the CGaAs PowerPC Microprocessor was formally proposed based upon these findings.

CHAPTER 6

FXU ARCHITECTURE OVERVIEW

Chapter 4 presented the proposed PUMA system and defined the functionality that would be required of the CGaAs PowerPC microprocessor. Chapter 5 detailed the optimization of a microarchitecture suitable for implementing a superscalar PowerPC microprocessor in the CGaAs technology. In this chapter, the FXU microarchitecture is formally specified, along with a description of the microprocessor pipeline.

6.1 Processor Organization

The microarchitecture of the FXU is based upon the extra-small microprocessor configuration defined in Section 5.3.7. A block diagram of the FXU is shown in Figure 6.1. The 1 KB on-chip instruction cache (L1-Icache) is supplemented by a two-entry stream buffer (Stream). The instruction fetch mechanism (Fetch) is guided by a 256-Byte two-level dynamic branch predictor (BP) implementing global sharing and a 64-entry Branch Target Buffer (BTB). The 16 KB primary data cache (L1-Dcache) is located off-chip, presenting a three-cycle memory access latency. The primary data cache latency is overcome by a pipelined load-store unit (LSU), load and store enhancements, and the superscalar nature of the execution core. Both the Arithmetic Logic Unit (ALU) and the Branch Resolution Unit (BRU) have a single-cycle latency. The dual-issue out-of-order superscalar execution scheme requires only an eight-entry Reorder Buffer (ROB) and two

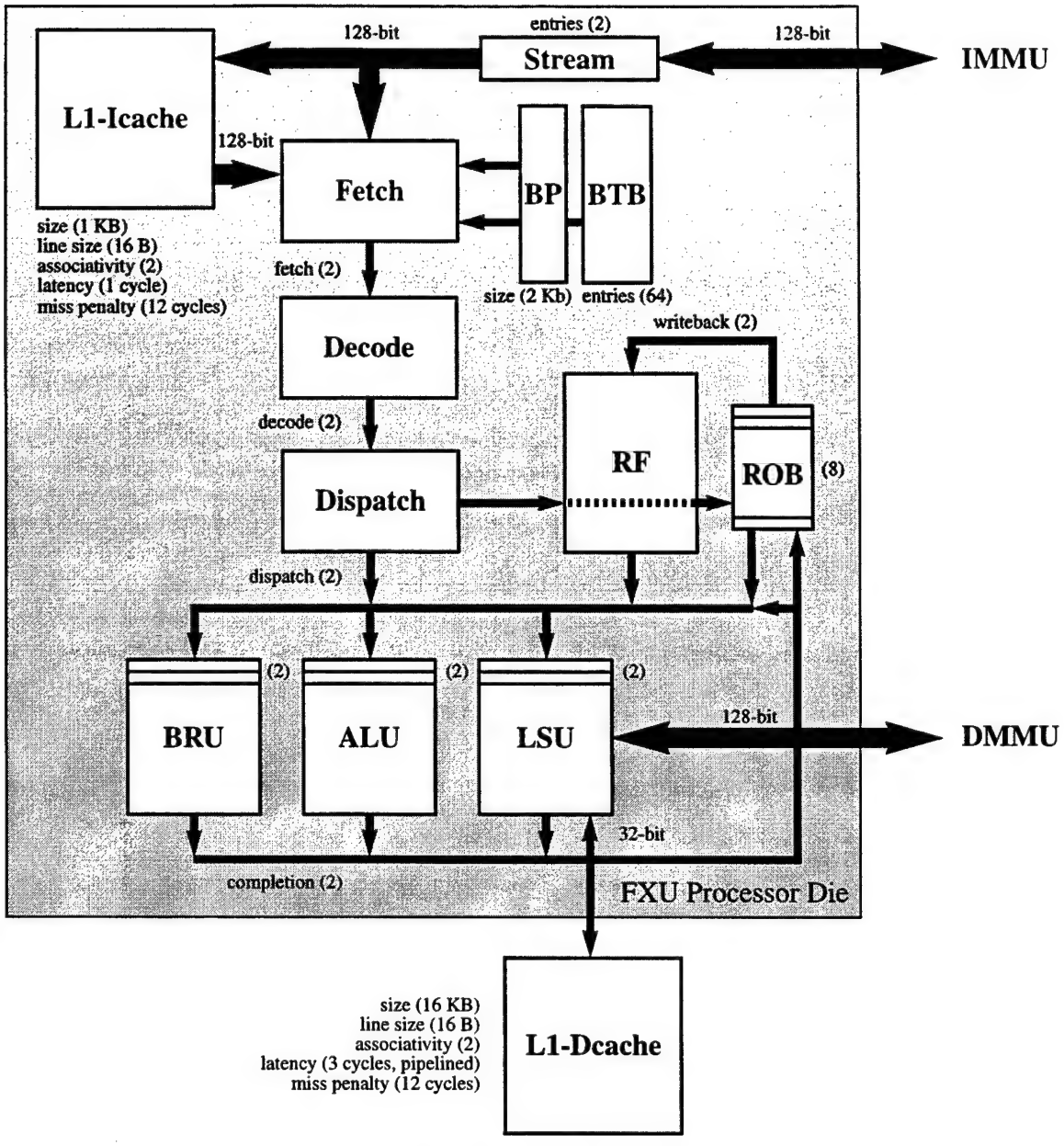


Figure 6.1: Processor block diagram.

reservation stations per functional unit. The widths of the internal buses, indicated in the figure, are such that the machine would ideally support an execution rate of two instructions per cycle (IPC). The connections to the off-chip memory management units (IMMU and DMMU) are equivalent to a full cache line, 16 B, or 128 bits. Simulations estimate that the processor will be able to sustain an execution rate of 0.76 IPC, translating to 153 MIPS at 200 MHz.

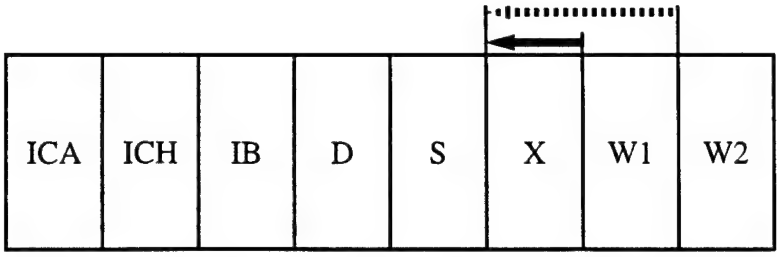


Figure 6.2: Logical view of the FXU pipeline.

6.2 Pipeline Overview

Perhaps the best way to describe the operation of the FXU microarchitecture is to describe the pipeline. A logical view of the pipeline is presented in Figure 6.2 and a functional description in Table 6.1. Under optimal operating conditions, the FXU has an eight-stage pipeline. Stall conditions, branch mispredictions, and load and store instructions will see a longer effective pipeline. Forwarding paths are indicated by the arrows. The solid arrow indicates results that are forwarded to the reservation stations directly from the completion bus. The dashed arrow indicates results that are forwarded through

Pipeline Stage		Functional Description
ICA	Instruction Cache Access	Access primary instruction cache and branch predictor. Form next program counter value.
ICH	Instruction Cache Hit	Determine instruction cache and stream buffer hit status. Select between instruction cache and stream buffer data.
IB	Instruction Buffer	Align instruction words for the decoder.
D	Decode	Decode instructions.
S	Schedule	Access register file and reorder buffer to form operands. Route instruction and operands to the appropriate reservation stations.
X	Execute	Single-cycle (ALU and BRU) or multi-cycle (LSU) execution.
W1	Writeback 1	Forward results to dependent instructions in the reservation stations. Update lookahead state in the reorder buffer.
W2	Writeback 2	Update architectural state.

Table 6.1: Functional description of FXU pipeline stages.

the reorder buffer, to instructions as they are dispatched. The following subsections will discuss the functions and hardware associated with the individual pipeline stages.

6.3 Instruction Fetch

The first three stages of the FXU pipeline (see Figure 6.3), ICA, ICH, and IB, represent the fetch pipeline, similar to what was implemented in the Instruction Cache Unit (ICU) in the POWER implementations. The fetch pipeline supplies the processor core with a steady stream of instructions for execution. It must maintain the instruction stream despite branch instructions which often produce short instruction runs; almost half of all instruction runs consist of four instructions or less [28]. An efficient fetch mechanism must also deal with instruction cache misses, various stall conditions, and machine exceptions.

Instruction Cache Access	Instruction Cache Hit	Instruction Buffer
• Primary Instruction Cache Access • Branch predictor access • Next program counter select • Branch prediction taken • BTB target address select • Branch predictor BHR speculative update	• Primary instruction cache and stream buffer hit / miss status • Instruction cache line select • Initiate next stream access	• Align instructions for decoder • Update pointer into current cache line • Advance instruction buffer

Figure 6.3: Functional description of the instruction fetch pipeline.

The fetch mechanism integrates the fetch control logic, primary instruction cache, stream buffer, branch predictor, branch target buffer, and instruction buffer. A block diagram of the fetch mechanism is shown in Figure 6.4. The BTB [50,51] caches the target address for taken branch instructions. The BTB has 64 entries, corresponding to the number of lines in the instruction cache. This allows one taken branch per cache line to store a target address in the BTB. Together the two-level dynamic branch predictor and the BTB form the basis for an efficient fetch mechanism, similar to those discussed in [52,53,54].

The fetch logic represents the centralized control for all fetch mechanism resources in the ICA stage of the pipeline. The most critical function of the fetch logic is program counter maintenance. The fetch module interfaces with the writeback module to identify exceptional conditions which will redirect the dynamic instruction stream. Exceptions may be either architectural exceptions, such as PowerPC ISA defined exceptions, or microarchitectural exceptions, such as branch mispredictions. Resource hazards further

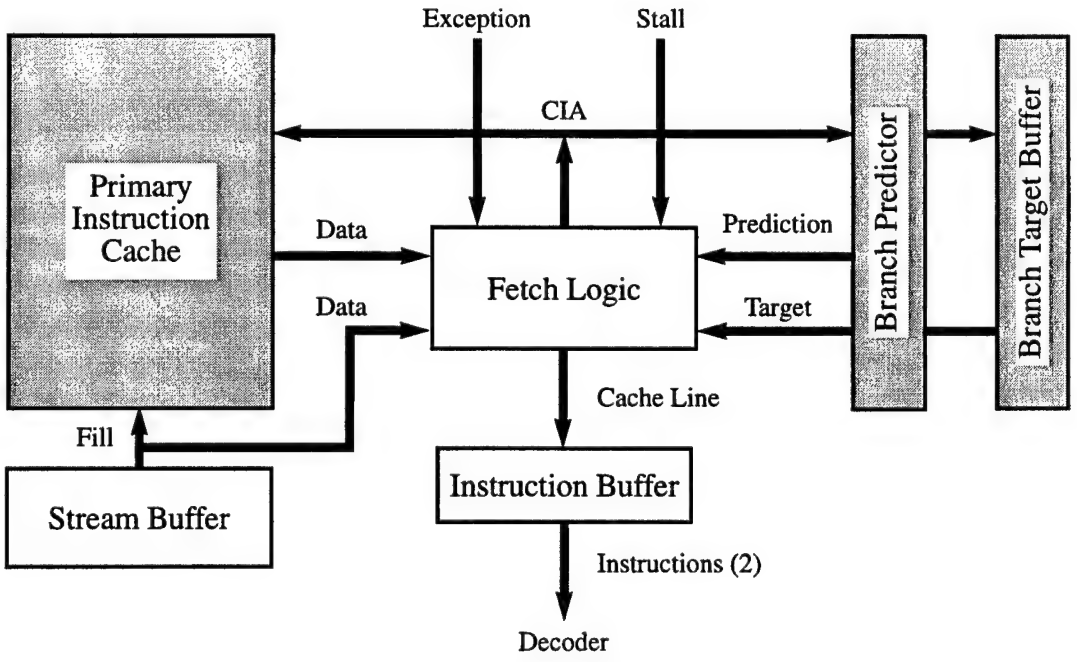


Figure 6.4: Fetch mechanism block diagram.

down the pipeline may stall the dynamic instruction stream. The branch predictor and branch target buffer deliver branch prediction information and target addresses to the fetch module. If a branch is predicted taken, the target address is selected as the next speculative program counter value. When no exceptional conditions exist, the fetch module increments the program counter to the next sequential cache line. The current program counter (also referred to as the current instruction address, or CIA, in the PowerPC ISA description) is used to access the primary instruction cache, branch predictor, and branch target buffer.

The output of the instruction cache is latched at the end of the ICA stage, eliminating a potential critical path. In the ICH stage, the latched instruction cache tag and stream buffer tag are compared with the program counter to determine hit and miss status. If the access hit in the instruction cache, the data is transferred to the instruction buffer. If the access missed in the instruction cache and hit in the stream buffer, the stream buffer data is routed to the instruction buffer, and simultaneously written into the primary instruction cache. Writing the data into the cache is intended to satisfy the principle of temporal locality, allowing future references to the same data to hit in the instruction cache. However, since the instruction cache is single ported, the instruction cache write introduces a bubble into the fetch pipeline, requiring the read access in the ICA stage to stall for one cycle. A hit in the stream buffer also generates a memory request for the next sequential line in the stream. If the access is found to miss in both the stream buffer and primary instruction cache, a new stream will be initiated and the fetch mechanism stalls until the data arrives. The end result of the activity performed in the ICH stage is that a new instruction cache line is loaded into the instruction buffer and the appropriate prefetch access is initiated.

The stream buffer implements the resume policy [55], rendering the resource as non-blocking, permitting execution of correct-path instructions while servicing a wrong-path instruction cache miss.

The third stage of the fetch pipeline is occupied solely by the instruction buffer. The instruction buffer is a combination of flip-flops and multiplexors, specifically designed to select instructions for delivery to the processor core. The instruction buffer consists of two entries, each capable of holding an entire cache line, allowing a total of eight instructions to be buffered. After the instruction buffer has finished issuing instructions from one cache line, the instruction buffer advances and is able to accept a new cache line. The instruction buffering alleviates some of the instruction cache miss penalty and bubbles that would be introduced during the stream buffer to the instruction cache copy procedure.

6.4 Instruction Decode

The decode unit translates PowerPC instructions into unit-operations to be processed by the execution core. Recall from the previous chapter, that the philosophy behind unit-operations is to balance cycle time, while simplifying dependency resolution and write-back. Some of the PowerPC instructions must be translated into a sequence of unit-operations. A vector decoder design, shown in Figure 6.5, allows the decode unit to sequence through the translation process. The primary decoder (idecode) translates the PowerPC instruction into the first unit-operation and an index (uindex) to the next unit-operation in the sequence. The secondary decoder (udecode) uses the original PowerPC instruction and the index to produce subsequent unit-operations. When the secondary decoder has finished translating the PowerPC instruction, control is returned to the primary decoder to begin

translating the next PowerPC instruction. The vector decoder is capable of producing one unit-operation per cycle. The decode unit also employs a second vector decoder. This allows the decode unit to produce the two unit-operations per cycle to support the dual-issue superscalar core. The decode unit is able to decode two simple PowerPC instructions per cycle. When a compound PowerPC instruction is encountered, the second vector decoder is disabled and the translation proceeds from the first vector decoder in a serialized manner. The performance impact of this limited decode policy is minor, given that the low occurrence of compound PowerPC instructions.

The translation process can be as simple as mapping a PowerPC instruction to a single unit-operation, or it may be far more complicated. The PowerPC "rotate left word immediate then mask insert and record" instruction, rlwimi., results in the most complicated

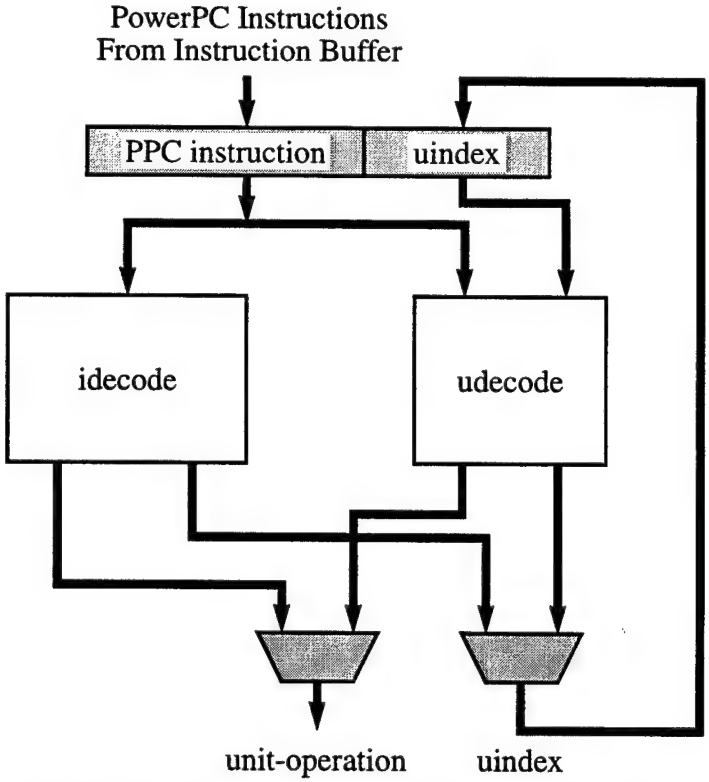


Figure 6.5: Block diagram of FXU decode unit.

sequence of unit-operations; Table 6.2 details the translation process for this instruction. The destination and source registers that begin with an "x" are temporary physical registers not visible to the overlying ISA. Though the sequence spans six unit-operations, the execution of the sequence is atomic. Exception detection is suppressed until the entire PowerPC instruction has been completed.

uindex	next uindex	uop	functional unit	destination register	immediate value	source a register	source b register
-	0xC	rli	alu	x0	SH	rS	-
0xC	0xD	mask	alu	x1	MB,ME	-	-
0xD	0xE	and	alu	x2	-	x0	x1
0xE	0xF	andc	alu	x3	-	rA	x1
0xF	0x0	or	alu	rA	-	x2	x3
0x0	-	cmpi	alu	CR0	0	rA	-

Table 6.2: Translation of a compound PowerPC instruction.

6.5 Scheduling

The fifth stage of the processor pipeline, referred to as the scheduling stage, S, is responsible for routing unit-operations and the required operands to the reservations stations in the execution core. This process is known as instruction scheduling, or instruction dispatch. The decoder specifies the required operands by producing an architectural register index corresponding to each source operand. The dispatch module uses the index to access the register file and the reorder buffer. The output of the register file reflects the most recent value of the architectural register that has been committed to architectural state. The reorder buffer, on the other hand, maintains the lookahead state of the architectural registers in a set of physical registers. If the reorder buffer detects that there is a lookahead value for the operand register, it will return either the value, if available, or a tag corresponding to the result. The reservation stations use the tag to obtain the value as results are forwarded over a common data bus [42] after execution. The dispatch module routes the register file values, reorder buffer values, or physical register tags to the reservation stations in the execution core.

6.6 Execution

The execution core consists of the functional units and their associated reservation stations. The set of unit-operations is divided into three functional categories, arithmetic operations, branch operations, and memory operations, which are implemented as an Arithmetic Logic Unit (ALU), a Branch Resolution Unit (BRU), and a Load-Store Unit (LSU). Attached to each function unit is a set of reservation stations and a result register.

A distributed reservation station scheme is implemented in the execution core with two reservation station entries attached to each functional unit. Instructions are scheduled into the reservation stations from the dispatch bus. When the necessary operands are available, the instruction proceeds from the reservation station to the functional unit execution logic; this process is referred to as instruction issue. The reservation stations provide a buffer between instruction schedule and instruction issue, allowing the dispatcher to schedule additional instructions for execution while the reservation stations and functional units work to resolve dependencies.

6.6.1 Functional Unit Pipeline

The ALU and BRU perform single-cycle execution of arithmetic and branch instructions. The LSU is a multi-cycle unit designed to interface with an off-chip primary data cache. The objective of the LSU is to provide a throughput of one memory access per cycle, so the off-chip cache will have approximately the same performance as that of an on-chip cache. The pipeline of the LSU is shown in Figure 6.6. When an instruction is issued from the reservation stations, the address generation unit computes the effective address (EA) of the memory access. The instruction and EA are then latched by the load access latch or store access latch. In the second stage loads are allowed to bypass stores, the EA is transmitted to the off-chip data cache, and the instruction information is pushed onto the load-store access queue (LSAQ). Instructions are held in the LSAQ until the cache access is performed and the data is transmitted to the LSAQ. Once the data is received, the access is checked to determine the hit / miss status. If the access hit, the result is sent to the result register, otherwise the instruction is loaded into the miss status

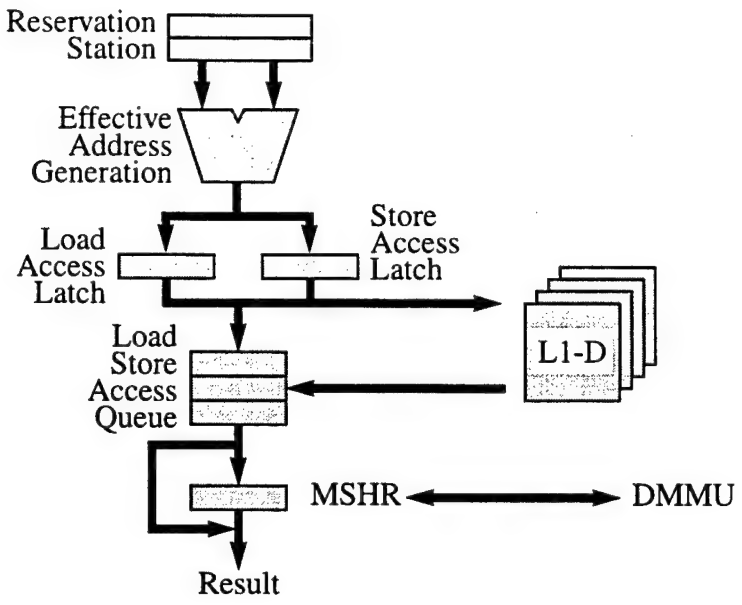


Figure 6.6: Load-store unit block diagram.

holding register [56] (MSHR) and the secondary cache memory is accessed. The MSHR enables the LSU to complete additional memory accesses while the cache miss is being serviced.

In addition to the hardware enhancements aimed at improving the performance of the LSU, software prefetch instructions are also incorporated [57,58,59]. Prefetch and post-store instructions, defined in the PowerPC ISA, were implemented in the FXU to allow the compiler to efficiently manage the data cache. The compiler can schedule a prefetch instruction well in advance of the actual load operation. The LSU then copies the data from the secondary cache into the primary data cache, allowing subsequent load operations to hit. If the prefetch operation misses in the secondary cache, no further action is taken and the load will inevitably miss. Post-store instructions cause the cache line to be written to main memory and the cache entry invalidated. Since the primary cache implements a write-through policy, flush instructions simply invalidate the primary cache line.

The secondary cache implements a writeback policy; requiring the MMU to copy modified lines to main memory while invalidating the entry.

6.7 Writeback

A result register at the end of each functional unit decouples execution from writeback. The primary advantage of this scheme is cycle time savings. Without result registers, the result forwarding and writeback delay would be added to the execution logic delay, creating a potential critical path. The writeback pipeline begins with the result registers and is divided into two stages. During the first stage, W1, results are forwarded to dependent instructions in the reservation stations and written into the reorder buffer. During W2, exception conditions are identified and architectural state is updated. All instructions preceding a mispredicted branch must complete W2 before a branch misprediction exception is signalled. Johnson [28] showed that such a policy resulted in only a slight performance degradation, while greatly simplifying the hardware. Architectural state is updated by committing the contents of a physical register to the corresponding architectural register. A delay of one or more cycles can occur between the two stages of writeback. However, this does not compromise performance because once the result has been written into the reorder buffer, it is held in a physical register and can be forwarded to subsequent instructions as they are scheduled into reservation stations.

6.8 Reorder Buffer

The reorder buffer, shown in Figure 6.7, plays an important role in the scheduling and writeback stages. It also holds the status of all in-flight instructions and register mappings. The register mappings are kept by the register alias table (RAT). As instructions are sched-

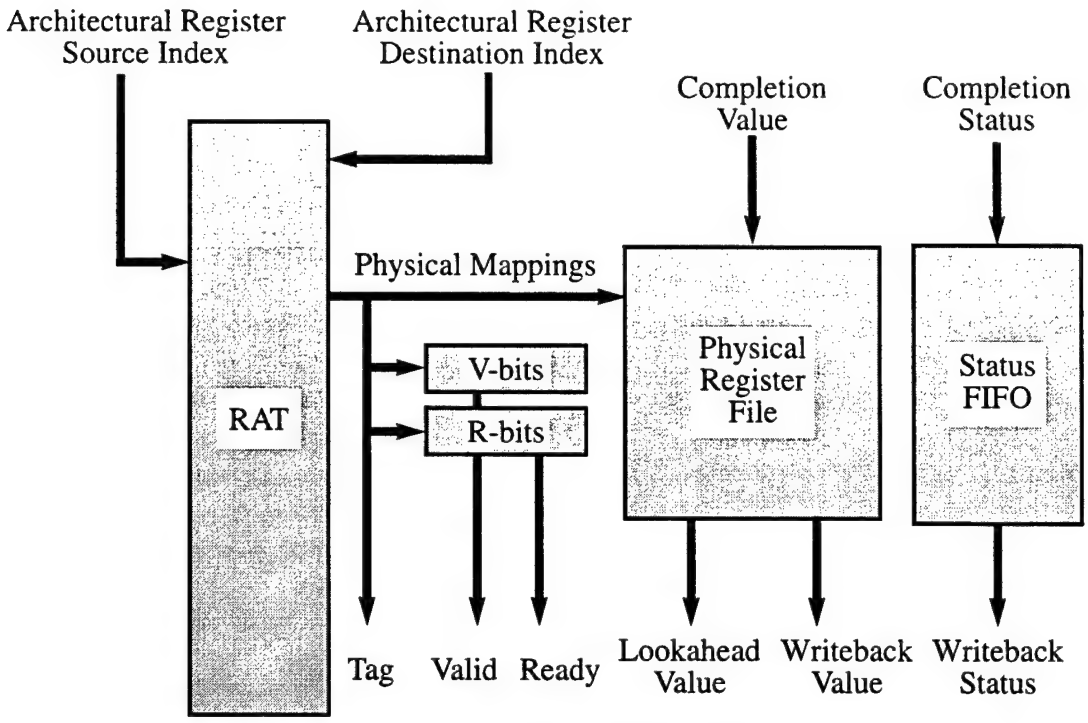


Figure 6.7: Reorder buffer block diagram.

uled, the reorder buffer maps the architectural register destinations to physical registers. During W1, the results of execution are written into the physical register file. During S, the architectural register indices of the source operands are applied to the RAT. If the reorder buffer has mapped the architectural registers to physical registers then the RAT returns the valid aliases (Physical Mappings). Each physical mapping is used to index into the physical register file and obtain the lookahead register value as well as the associated valid and ready bits. The final component of the reorder buffer is the status fifo. During S, information regarding the state of the processor and the particular instruction is written into the top of the status fifo. During W2, the status and result information is analyzed to determine whether an exceptional condition exists.

6.9 Summary

This chapter formally proposed the architecture for the CGaAs PowerPC microprocessor. The pipeline consists of eight stages. A fetch mechanism integrates a 1 KB primary instruction cache, 256 B two-level dynamic global sharing branch predictor, 64-entry BTB, and a two-entry stream buffer. The fetch mechanism is able to deliver two instructions per cycle to the dual-issue superscalar core. The decode mechanism is able to decode two simple PowerPC instructions per cycle, or translate one compound PowerPC instruction into a series of unit-operations over the course of multiple cycles. The dual-issue superscalar core is maintained by an eight-entry reorder buffer and two reservation stations per functional unit. The pipelined LSU interfaces with the off-chip primary data cache to maintain a constant throughput of one memory operation per cycle.

CHAPTER 7

PUMA MICROPROCESSOR IMPLEMENTATION

Central to the design of the PUMA system is a CGaAs PowerPC microprocessor. The previous chapters have optimized a superscalar microarchitecture suitable for the CGaAs technology and formally specified the operation of the pipeline. In this chapter we focus on the implementation details of the PowerPC microprocessor. The microprocessor was implemented in both CMOS and CGaAs technologies. The 0.35 μm CMOS process from Taiwan Semiconductor Manufacturing Company (TSMC) was made available through the MOSIS VLSI Fabrication Service. The initial version of the FXU, dubbed FXU I, implemented in the TSMC process, was intended to serve as a logic debug vehicle for the subsequent CGaAs FXU. This was done because complementary CMOS circuits are well studied and can be implemented reliably, giving FXU I a good chance at successfully demonstrating the functionality of the architecture. The subsequent CGaAs version of the FXU, referred to as FXU II, was designed in Motorola's 0.5 μm CGaAs process. Compared to complementary CMOS circuit design, digital CGaAs circuits are not as well characterized and represent a higher degree of risk. The following sections discuss implementation details pertaining to FXU I and FXU II.

7.1 FXU I Implementation Details

The FXU I microprocessor was designed using the EPOCH physical design environment [60], from Duet Technologies, taking advantage of the standard cell library and precompiled datapath macro-cells for the CMOS TSMC process. Customizing the EPOCH design environment to suit the needs of the CGaAs process was estimated to require a significant effort. The design of FXU I allowed the project to continue towards the goal of creating a CGaAs microprocessor while issues related to the CGaAs design environment were still being investigated.

7.1.1 Modifications to the FXU Architecture

Overall project schedule constraints required that the microprocessor development time be reduced. The most obvious solution was to simplify the architecture. Several modifications were made to the proposed microarchitecture to alleviate some of the complexity involved in designing a 32-bit superscalar microprocessor. The modifications succeeded in reducing the complexity, shortening the design cycle, and increasing the likelihood of producing a functional processor, but these gains came at the expense of overall performance.

A prototype register file for the CGaAs microprocessor had been previously developed. This register file had two read ports and one write port, while the proposed superscalar microarchitecture was dual issue, requiring a register file with four read ports and two write ports. We decided that a new register file would not be developed and the CGaAs FXU would be a one-wide out-of-order machine. Following this crucial decision, it made sense that FXU I be single-issue as well. This decision simplified the scheduling, register file, reorder buffer, and writeback logic. While FXU I is single issue, it still maintains a

three-wide out-of-order execution core to compensate for the increased primary data cache latency. Such a processor has been shown to achieve about 60% of the maximum performance [61]. Table 7.1 compares the performance of the single-issue FXU I versus the proposed dual-issue microprocessor. Performance is reduced by 18%, but FXU I still provides a slight improvement over a comparable sequential pipelined microprocessor.

Testability issues, rather than architectural complexity, motivated further simplification. Since FXU I was intended to serve as a logic demonstration prototype, MCM capabilities were deemed non-essential to the design. The goal of FXU I was to demonstrate the functionality of the architecture on an HP82000 digital IC tester, not as part of a system. The MCM design called for separate instruction and data memory buses to the MMUs and secondary memory system. Implementing the dual bus structure would have required hundreds of extra pins and presented packaging and testing difficulties. Instead, a unified bus structure was implemented. Access to the unified bus interface is determined by internal arbitration between the instruction and data streams, with priority given to data references. The unified-bus structure implements elements of the original dual-bus design, such as split transaction protocol and transaction pipelining. The proposed architecture also specified a 16 KB off-chip primary data cache. Given the high integration level of the CMOS process, and the need to further reduce pin count, a 4 KB data cache was implemented on-chip. The on-chip data cache still maintains a 3-cycle latency, so that the interface is identical to that of the CGaAs FXU design. These modifications reduced the number of pins required from 622 to only 204, a reduction of nearly 67%.

	FXU I	Model
Dispatch (instructions per cycle)	1	2
Register file ports (read/write)	2/1	4/2
Completion (instructions per cycle)	1	2
Data cache integration	on-chip	off-chip
Data cache size (KB)	4	16
Data cache latency (cycles)	3	3
Theoretical maximum IPC	1	2
Estimated transistor count (K)	800	500
IPC	0.6243	0.7642
Advantage over sequential pipeline	1.0291	1.2749

Table 7.1: Differences between FXU I and the proposed microarchitecture.

7.1.2 TSMC 0.35 μm CMOS Technology Summary

The EPOCH framework provided a design environment complete with a standard cell library, datapath macrocells, and memory structures designed in the TSMC 0.35 μm CMOS technology. The following sub-sections analyze the performance capabilities of the technology, logic gate sizing, and the standard cell library.

7.1.2.1 Estimating Gate Delay

HSPICE simulations of ring oscillators can be used as a coarse estimate of the delay of typical logic gates. Figure 7.1 shows the HSPICE simulation of a 31-stage ring oscillator comprised of minimum-sized inverters. The oscillator frequency is 211.5 MHz, corre-

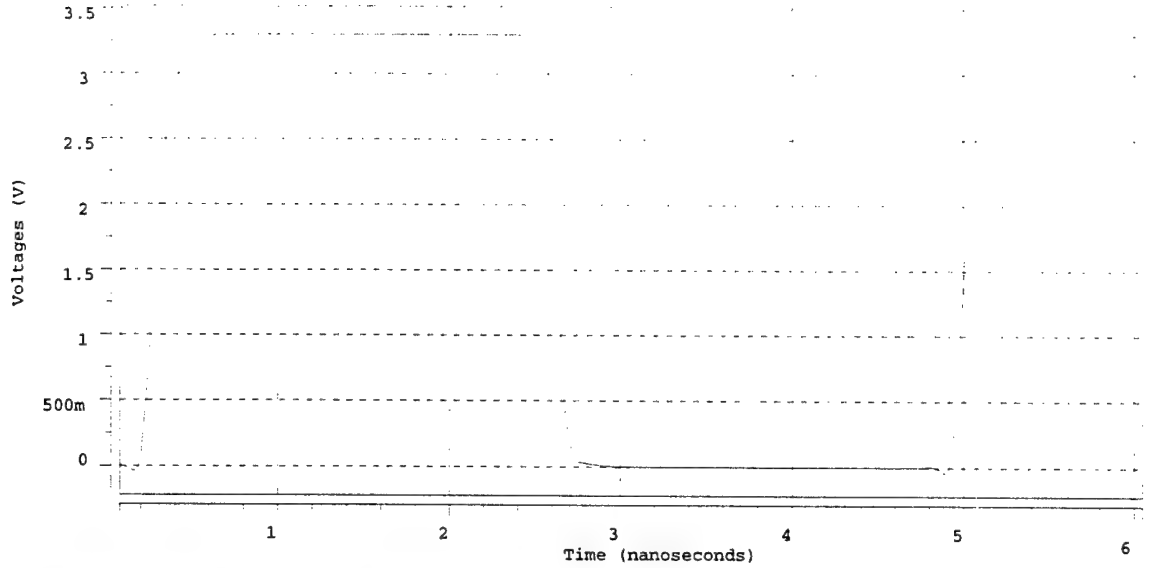


Figure 7.1: 31-stage CMOS inverter ring oscillator simulated at 211.5 MHz.

sponding to an average gate delay of 76.3 psec. The circuit consumes 1.202 mW, resulting in a power dissipation of 0.18 μW/MHz/gate.

Figure 7.2 shows the HSPICE simulation of a 31-stage ring oscillator consisting of minimum-sized 2-input NAND gates. The oscillator has a frequency of 143.2 MHz and

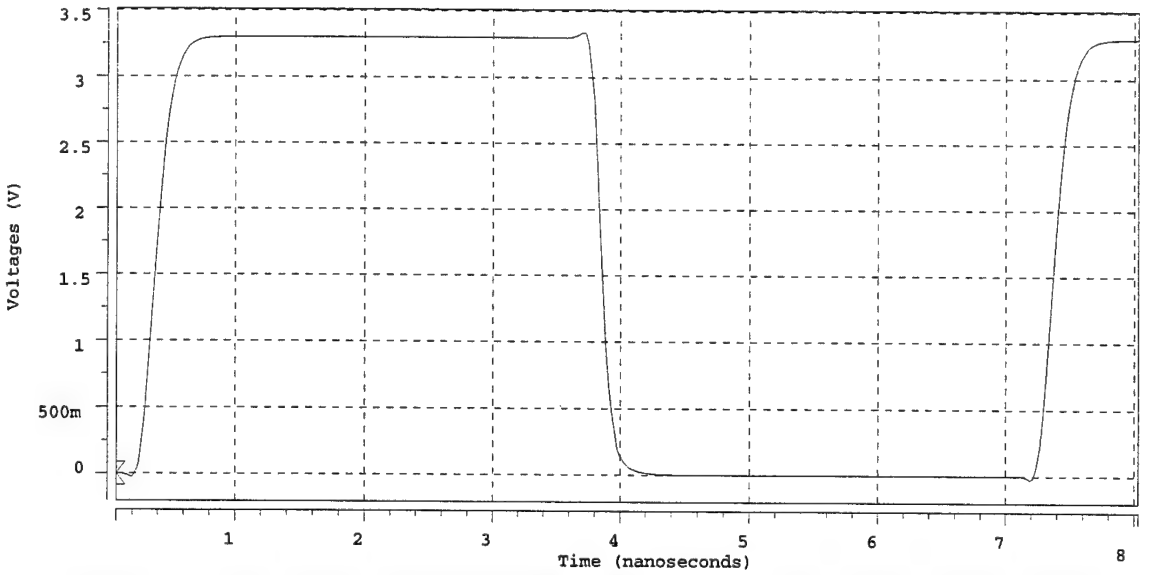


Figure 7.2: 31-stage CMOS NAND2 ring oscillator simulated at 112.6 MHz.

consumes 1.313 mW. The NAND2 gate has a typical delay of 112.6 psec and a power dissipation value of 0.30 µW/MHz/gate.

These HSPICE simulation results do not accurately reflect the performance of similar logic gates when instantiated in a microprocessor design. The inverter and NAND2 logic gates in the ring oscillators are not driving appreciable capacitive loads; the results represent the delay of the logic gates with a fan-out of one. However, ring oscillator frequency will be a convenient metric for a first-order comparison between CMOS and CGaAs.

7.1.2.2 Gate Sizing

The EPOCH standard cell library provides logic gates with varying drive strengths. An automated sizing tool allows logic gates to be properly sized for a given load. Figure 7.3 shows HSPICE simulations of a minimum-sized NAND2 logic gate (W_p/W_n = 1.625µm/1.675µm) driving loads ranging from 100fF to 500 fF. Table 7.2 lists the rise and fall times

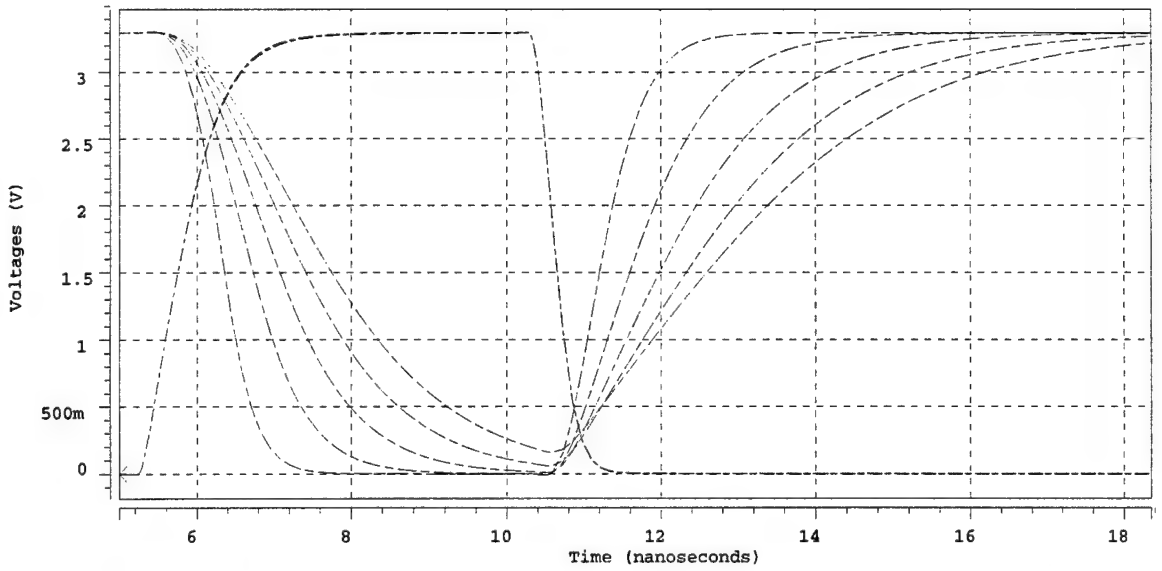


Figure 7.3: Minimum-sized CMOS NAND2 driving various loads.

C_{load} (fF)	t_r (nsec)	t_f (nsec)	t_{plh} (nsec)	t_{phl} (nsec)
100	1.135	0.946	0.623	0.518
200	2.088	1.564	1.059	0.864
300	3.057	2.189	1.489	1.183
400	4.036	2.821	1.882	1.491
500	5.023	3.458	2.193	1.795

Table 7.2: Characteristics of minimum-sized CMOS NAND2 gate.

along with the rising and falling propagation delays. Notice that in going from a 100 fF load to a 500 fF load, the rising and falling propagation delays have tripled, while the rise and fall times have quadrupled.

The width of the transistors can be increased, resulting in a greater drain current. Increased drain current will allow the circuit to charge and discharge the load capacitance more rapidly, thus the circuits can be properly sized for a given load. Figure 7.4 and Table 7.3 present the results of HSPICE simulations for NAND2 logic gates that have been appropriately sized for a given load. The loads again range from 100 fF to 500 fF while the drive strength varies from one to five. The drive strength is derived from the size of the output transistors; a gate with a drive strength of *n* has output transistors that are approximately *n* times as wide as a minimum-sized NAND2 circuit (W_p/W_n = 1.625μm/ 1.675μm). From Figure 7.4 and Table 7.3 we see that nearly identical slew rates and propagation delays can be maintained by sizing the logic gates appropriately.

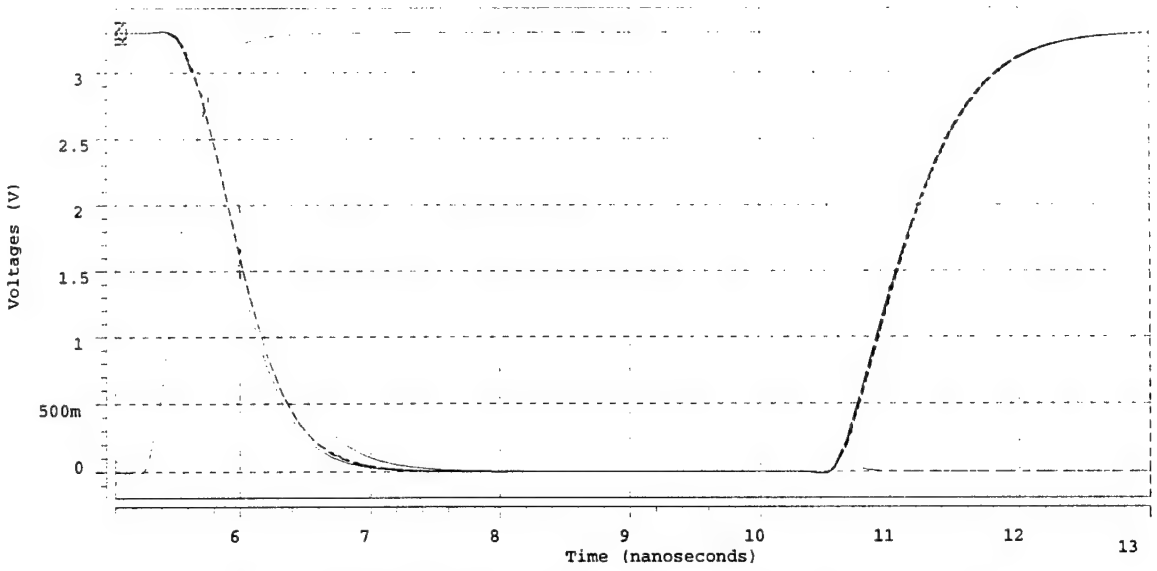


Figure 7.4: CMOS NAND2 gates sized for a given load.

Drive Strength	W_p/W_n (μm)	C_{load} (fF)	t_r (nsec)	t_f (nsec)	t_{plh} (nsec)	t_{phl} (nsec)
1x	1.6/1.7	100	1.090	0.799	0.549	0.432
2x	3.3/3.4	200	1.101	0.830	0.552	0.447
3x	4.9/5.1	300	1.085	0.815	0.548	0.441
4x	6.5/6.7	400	1.095	0.833	0.554	0.449
5x	8.1/7.1	500	1.097	0.956	0.552	0.518

Table 7.3: Characteristics of sized CMOS NAND2 gates.

Additional HSPICE simulations for fanout of four (FO4) configurations demonstrate that the family of NAND logic gates achieves nearly identical performance for every drive strength. The customized EPOCH environment for the CGaAs microprocessor development must also implement an automated means to size logic gates and achieve similar performance improvements.

7.1.2.3 Standard Cell Library

Table 7.4 summarizes the characteristics of some common elements of EPOCH's standard cell library for the TSMC process. The table lists the width and area of the standard cells, the standard cell height is fixed at 14.925 μm. The power and delay values were measured from HSPICE simulations with unloaded logic gates; these values therefore represent the intrinsic properties of the gates. Notice the balance between the NAND and NOR logic styles; the NAND2 and NOR2 have almost identical area, power, and delay values. The NAND2 consists of two n-type transistors in series, while the NOR2 consists of two p-type transistors in series. The balance between complementary transistors is a key advantage of CMOS over CGaAs. In CMOS the mobility of electrons is twice that of holes, whereas in CGaAs the mobility of electrons is four times that of holes. This causes a large discrepancy between the performance of n-type and p-type transistors in CGaAs, and consequently a large variation in the performance of the NAND and NOR logic families.

gate	width(μm)	area(μm^2)	Power(μW)	t_{delay}(psec)
AND2	7.20	107	177	174
INV	4.50	67	103	65
MUX2	11.25	168	319	221
NAND2	5.85	87	109	82
NOR2	5.85	87	103	88
OR2	7.20	107	173	179
XNOR2	9.90	148	219	227
XOR2	9.90	148	149	138

Table 7.4: TSMC 0.35 μm CMOS standard cell characteristics.

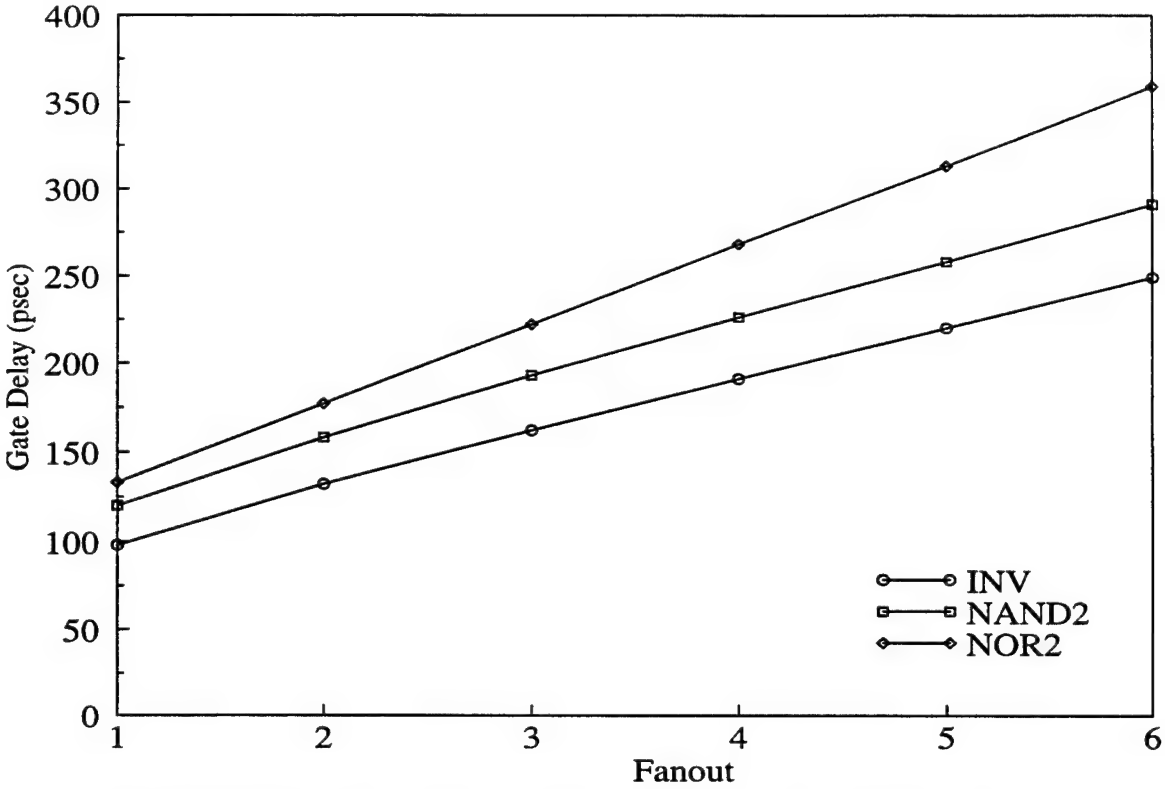


Figure 7.5: Fanout sensitivity of complementary CMOS circuits.

Figure 7.5 demonstrates the fanout sensitivity of complementary CMOS circuits based on HSPICE simulations. By knowing the intrinsic delay and the fanout sensitivity (see Table 7.5) it is possible to predict the delay of a gate with a fanout of n. Such plots facilitate comparisons of the fanout sensitivity of different circuit techniques and technologies. This data will allow a comparison of the fanout sensitivities of CMOS and CGaAs complementary circuits.

	INV	NAND2	NOR2
Intrinsic delay (psec)	71	89	87
Fanout sensitivity (psec/load)	30	34	45

Table 7.5: Delay and fanout sensitivity of complementary CMOS circuits.

7.1.3 Module Design Summary

Table 7.6 summarizes the fourteen modules that make up FXU I. The largest module is the execution cluster; comprised of three smaller sub-modules, the ALU, BRU, and LSU, along with the associated reservation stations. The next largest module is the 4KB on-chip data cache. The reorder buffer is also a fairly large module. It holds eight physical registers along with the architectural to physical register mappings used for register renaming. The FXU I core consists of 828,000 transistors, measures 7.5 x 6.4 mm, and was estimated through simulations to consume approximately 4 W.

Module	Dimensions width x height (μm)	Area (mm^2)	Transistor Count	Power Dissipation (mW)
Branch Predictor	661 x 689	0.456	21468	99
Branch Target Buffer	2035 x 705	1.437	28064	368
Instruction Fetch	1140 x 996	1.136	36715	155
Icache	1934 x 730	1.412	69934	428
Stream Buffer	1014 x 1080	1.097	20372	144
Decoder	769 x 1133	0.872	15734	85
Scheduler	561 x 1111	0.624	10132	75
Reorder Buffer	1913 x 1362	2.606	46890	352
Register File	1553 x 1300	2.021	52640	363
Execution Cluster	2735 x 3591	9.825	186928	696
Writeback	1312 x 430	0.566	8400	63
Dcache	2663 x 1949	5.193	284278	1034
D-bus Interface	1204 x 554	0.667	23594	108
I-bus Interface	743 x 919	0.684	20602	109
FXU I Core	7484 x 6395	47.869	827627	4092

Table 7.6: FXU I module statistics.

7.1.4 FXU I Performance Summary

TACTIC, a static timing analysis tool available within the EPOCH design environment, was used during full chip integration and optimization to identify critical paths. Most critical paths were optimized or redesigned, until a reasonable operating frequency was obtained. The next two sections summarize the static timing analysis of the final FXU I design.

7.1.4.1 Global Path Summary

A distribution of the paths in FXU I, as measured by gate delays, or logic levels, is presented in Figure 7.6. The percentage of the total number of paths is given rather than the raw counts. Most paths are less than 15 gates, but almost 10% of the paths in FXU I have 21 logic levels, or more. The long paths lie primarily in the execution units. However,

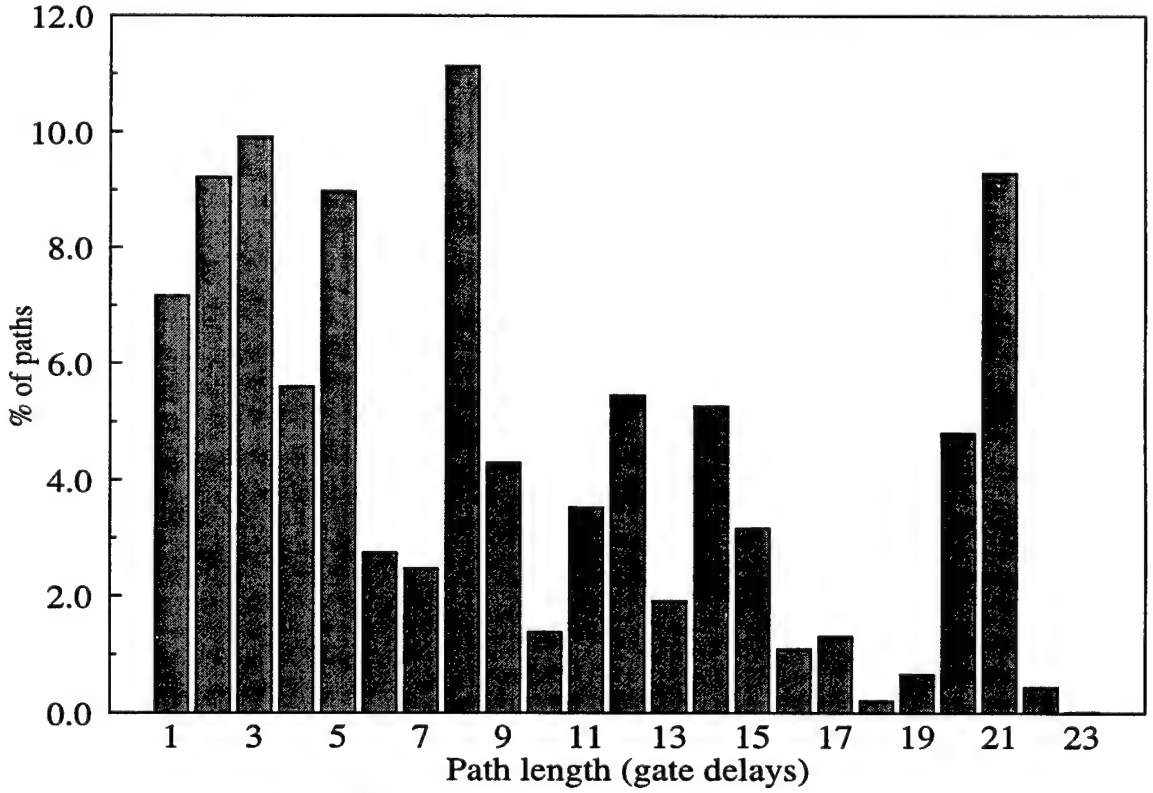


Figure 7.6: FXU I path distribution in terms of gate delays.

given the localized nature of the gates, these paths were not critical paths, in terms of actual delay, despite the large number of gate delays.

Figure 7.7 presents the distribution of paths in FXU I in terms of total delay measured in nanoseconds. Most paths are less than 10 nanoseconds, however a small number of paths have delays of almost 13 nanoseconds. These paths limit the operating frequency of the processor to approximately 80 MHz. Further analysis and optimization could have eliminated most of these paths, increasing the operating frequency of FXU I to 100 MHz. However, optimizing the operating frequency was not the primary goal of FXU I.

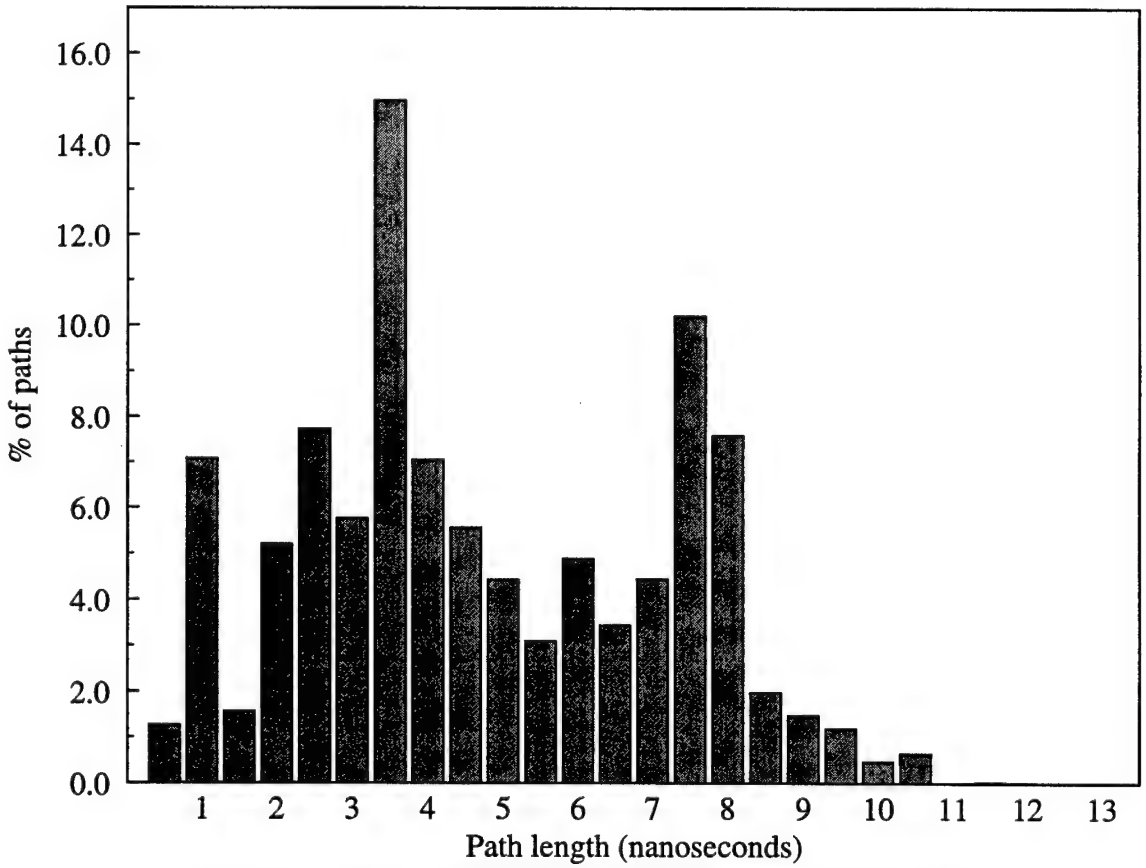


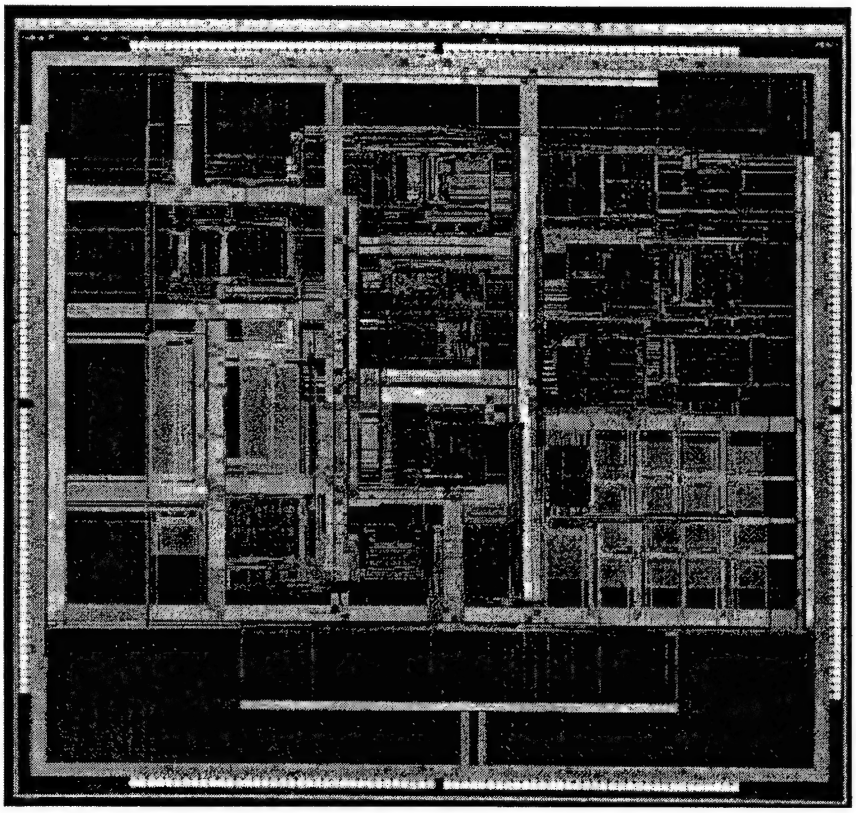
Figure 7.7: FXU I path distribution in terms of total delay.

7.1.4.2 Critical Path Analysis

The operating frequency of FXU I is limited by a path through the branch predictor. The branch prediction mechanism is a two-level structure, consisting of a branch history register (BHR) and a pattern history table (PHT). In the first half of the processor clock cycle, the BHR and the program counter are exclusive-ored to form an index into the PHT. On the falling edge of the clock, the PHT read access is initiated and the value of the two-bit predictor is transmitted to the fetch module. The PHT read access requires 4.4 nanoseconds, and additional cycle time is consumed buffering the value of the two-bit counter to the fetch module. Once inside the fetch unit, the two-bit counter value must pass through an input multiplexor, before arriving at the input of a D flip-flop. Assuming the FXU I operates on a 50% duty cycle clock, this path fixes the FXU I cycle time at 12.511 nanoseconds. Several steps could have been taken to reduce the effects of this path. The PHT read could have been initiated earlier in the clock cycle, rather than during the second half of the cycle. Detailed analysis of the path revealed that the buffers driving the two-bit counter value from the PHT to the fetch unit are undersized. The buffer delay accounts for nearly 27% of the total delay; simply increasing the size of the output buffers would have significantly reduced the delay of the path. Unfortunately, this path was not discovered until after the final design was sent to fabrication. Optimization of this and other critical paths within FXU I was not a priority for the design team. FXU I was intended to serve as a prototype for the CGaAs implementation of FXU II.

7.1.4.3 FXU I Design Statistics

A die photo of FXU I and a summary of the key design statistics are shown in Figure 7.8. The core is considerably smaller than the die, occupying only about half of the total die area. This inefficiency is a result of the unified packaging solution adopted for the CMOS FXU I and the CGaAs FXU II. The same die footprint and package were selected for both microprocessors to ease packaging and testing. The 0.5 μm CGaAs design rules are much coarser than those of the TSMC 0.35 μm CMOS process, so it was predicted that the CGaAs die would be significantly larger. A 391-pin ceramic PGA from Kyocera, PB-P90170-A (P/N 32639101), was selected as a suitable package for both designs. The dimensions of FXU I periphery were stretched to meet the bonding rules for the package.



- 32-bit PowerPC single-issue 3-way superscalar
- 4 KB Dcache / 1 KB Icache
- 0.35μm TSMC CMOS, 3-metal layers
- 829,533 transistors
- 9.9 x 9.9 mm die
- (7.5 x 6.8 mm core)
- 80 MHz
- 4.1 Watts @ 3.3 V
- 280 pins (220 I/O)
- 391-pin PGA
- Fully static
- Standard cells
- Datapath macrocells

Figure 7.8: FXU I die photo and design statistics.

7.2 FXU II Implementation Details

Following the completion of the FXU I design, attention was turned to the development of the CGaAs microprocessor, FXU II. For the most part, the design of FXU II was a remapping of FXU I onto a CGaAs standard cell library. Through discussions with the CGaAs foundry, the integration level for FXU II was set at 400,000 transistors. Significant modifications to the proposed FXU architecture, and the architecture of FXU I, were required in order to meet the limited transistor budget.

7.2.1 Modifications to the FXU I Architecture

Planned modifications to the FXU I architecture were to reduce the FXU II transistor count to an estimated 500,000 transistors. FXU II implements an off-chip 16 KB data cache, as opposed to the 4 KB on-chip data cache of FXU I. This modification did not require any functional changes, since a suitable interface to the off-chip data cache had been designed into the LSU of FXU I, emulating the interaction with an off-chip data cache. An array of area interconnect pads was developed to create a high-speed interconnection between FXU II and the SRAMs of the off-chip primary data cache. However, this architectural modification only reduced the transistor count to 550,000. Additional modifications were required to further reduce the implementation cost to 400,000 transistors. Table 7.7 summarizes the major architectural modifications made to FXU II and the corresponding reduction in transistor count. The design of FXU I is used as the starting point.

Modification	Transistor Count (thousands)
FXU I	830
Off-chip primary data cache	-284
Redesigned register file	+68
Elimination of branch prediction	-50
Elimination of superscalar execution	-87
Elimination of out-of-order execution	-40
Redesign of fetch mechanism	-54
FXU II	383

Table 7.7: Architectural modifications and transistor budget of FXU II.

Some of the advanced architectural features were eliminated in order to meet the integration requirement and ensure a functional prototype CGaAs microprocessor. Branch prediction, out-of-order execution, and superscalar execution were all eliminated, resulting in a savings of 177,000 transistors. Furthermore, the fetch mechanism was redesigned and instruction cache line prefetching removed; reducing the transistor count by 54,000. These modifications were necessary because the register file consumed an extra 68,000 transistors. The FXU I register file was based on 6-T SRAM cells. For FXU II, a more reliable D flip-flop design was chosen. Instead of a single 6-T SRAM cell, the redesigned register file requires a D flip-flop to store a single bit and two tristate buffers per bit to implement the two read ports. The net result of these modifications is an architecture for FXU II that requires only 383,000 transistors, but suffers considerably on the performance side. FXU II is capable of executing only 0.5 IPC, or only 83% of what a comparable pipelined microprocessor could achieve, yet requires more transistors. Here it is important to remember that FXU II is a demonstration of what can presently be accomplished in

CGaAs. FXU II incorporates many elements of a higher performance machine but is unable to afford the necessary buffer resources to implement the proposed superscalar microarchitecture.

7.2.2 Motorola 0.5 μm CGaAs Technology Summary

The 1.3V 0.5 μm 3-layer metal CGaAs process from Motorola served as the technology for FXU II. An automated design environment for the CGaAs technology did not exist, so the EPOCH system was customized[1] to create an automated CAD environment using the CGaAs process. The development of the standard cell library and a gate sizing methodology were created for the CGaAs technology. In this section, we take a brief look at the performance potential of the CGaAs technology and compare the results with the TSMC 0.35 μm technology utilized in the design of FXU I.

7.2.2.1 Estimating Gate Delay

Figure 7.9 shows the HSPICE simulation of a 31-stage ring oscillator comprised of minimum-sized inverters. The oscillator frequency is 102.5 MHz, corresponding to a typical inverter delay of 157 psec. The circuit consumes 0.511 mW, yielding a power-delay factor of 0.16 μW/MHz/gate.

1. The development of the customized EPOCH CGaAs physical design environment was led by Phiroze N. Parakh.

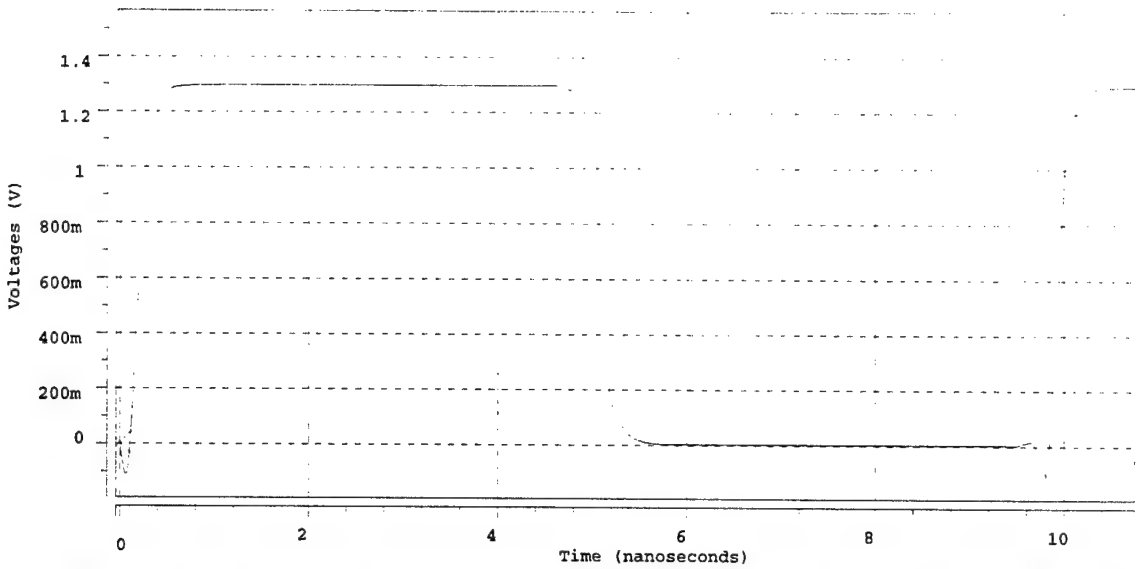


Figure 7.9: 31-stage CGaAs inverter ring oscillator simulated at 102.5MHz.

Figure 7.10 shows the HSPICE simulation of a 31-stage ring oscillator consisting of minimum-sized two-input NAND gates. The oscillator has a frequency of 79.9 MHz and consumes 0.587 mW. The NAND2 gate has a typical delay of 202 psec and a power dissipation of 0.237 μW/MHz/gate.

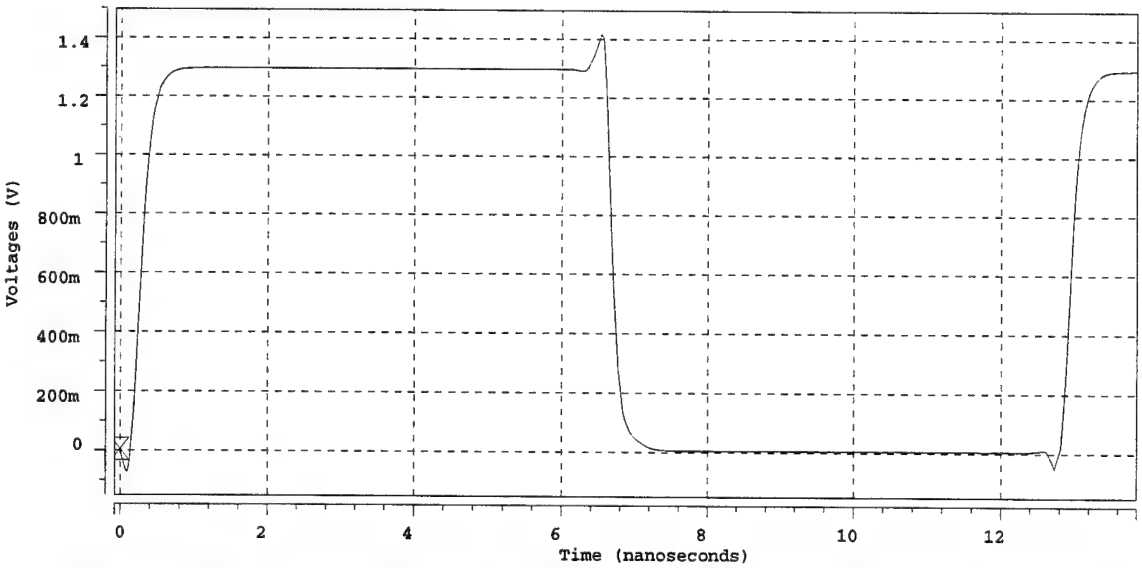


Figure 7.10: 31-stage CGaAs NAND2 ring oscillator simulated at 79.9 MHz.

A comparison of the various ring oscillators simulated in the 0.5 µm CGaAs and 0.35 µm CMOS technologies is presented in Table 7.8. The CMOS logic gates are faster than their CGaAs counterparts by a factor of two. However, due to the higher supply voltage, 3.3V versus 1.3V, and the faster operating frequency, the CMOS logic gates also consume more power than the CGaAs gates.

	0.5 µm CGaAs	0.35 µm CMOS
$f_{inv\text{-}ring}$ (MHz)	102.5	211.5
$P_{inv\text{-}ring}$ (mW)	0.511	1.202
$t_{inv\text{-}gate}$ (psec)	157	76
$P_{inv\text{-}gate}$ (µW/MHz/gate)	0.161	0.180
$f_{NAND2\text{-}ring}$ (MHz)	79.9	112.6
$P_{NAND2\text{-}ring}$ (mW)	0.587	1.313
$t_{NAND2\text{-}gate}$ (psec)	202	113
$P_{NAND2\text{-}gate}$ (µW/MHz/gate)	0.237	0.300

Table 7.8: Comparison of CGaAs and CMOS ring oscillators.

7.2.2.2 Gate Sizing

The CGaAs logic gates can be sized in a manner similar to the CMOS logic gates in Section 7.1.2.2. Figure 7.11 shows the HSPICE simulation results for a NAND2 logic gate driving loads ranging from 100fF to 500fF. The W/L ratio of the transistors is the same as those of the CMOS NAND2 circuit. The p-type transistor width is 2.3 μm and the n-type transistor width is 2.4 μm. Table 7.9 lists the rise and fall times along with the rising and falling propagation delays. Notice that in going from the smallest load to the largest load, the rising and falling propagation delays have tripled, while the rise and fall times have quadrupled.

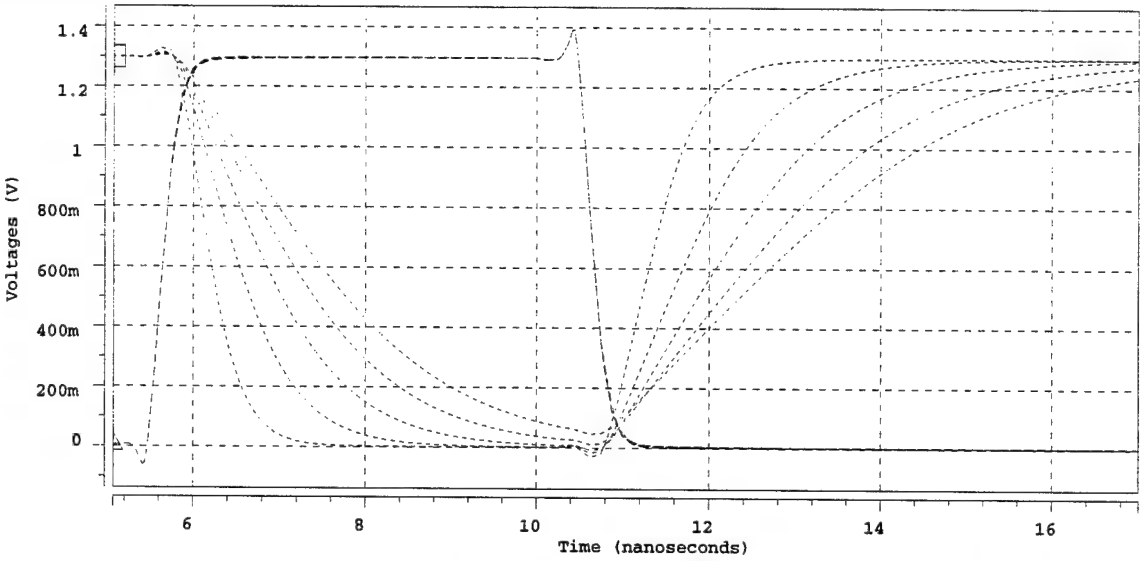


Figure 7.11: Minimum-sized CGaAs NAND2 driving various loads.

C_{load} (fF)	t_r (nsec)	t_f (nsec)	t_{plh} (nsec)	t_{phl} (nsec)
100	1.092	0.802	0.686	0.525
200	1.998	1.442	1.112	0.822
300	2.915	2.089	1.529	1.115
400	3.836	2.743	1.911	1.408
500	4.758	3.397	2.217	1.700

Table 7.9: Characteristics of minimum-sized CGaAs NAND2 gates.

An assortment of functionally equivalent logic gates with varying drive strengths offers a solution to the speed versus power optimization problem. Figure 7.12 shows the simulation results of NAND2 gates with drive strengths ranging from 1x to 5x driving capacitive loads of 100fF to 500fF. Table 7.10 summarizes the critical performance characteristics of the family of NAND2 logic gates. From the data we see that nearly identical transition times and propagation delays can be maintained by sizing the logic gates appropriately.

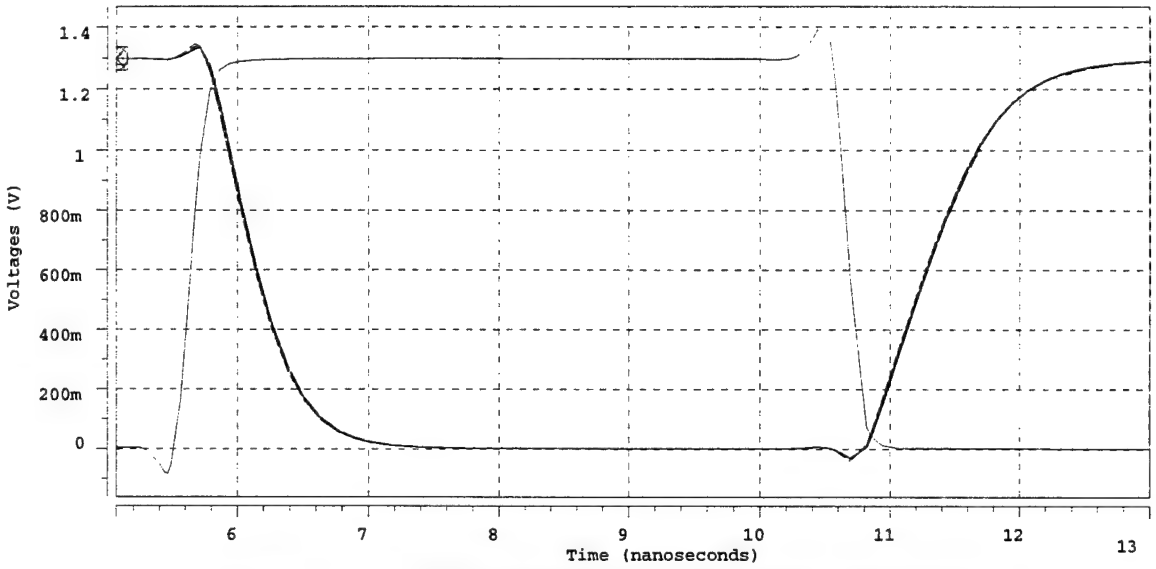


Figure 7.12: CGaAs NAND2 gates sized for a given load.

Drive Strength	W_p/W_n (μm)	C_{load} (fF)	t_r (nsec)	t_f (nsec)	t_{plh} (nsec)	t_{phl} (nsec)
1x	2.3/2.4	100	1.109	0.754	0.647	0.465
2x	4.6/4.8	200	1.074	0.733	0.627	0.453
3x	6.9/7.2	300	1.065	0.728	0.625	0.452
4x	9.2/9.6	400	1.064	0.727	0.626	0.454
5x	11.5/12.0	500	1.065	0.727	0.631	0.459

Table 7.10: Characteristics of sized CGaAs NAND2 gates.

The results of the CGaAs sizing experiment are nearly identical to those of the CMOS example in Section 7.1.2.2. The rise and fall times for the CGaAs NAND2 gates are slightly faster than those of the CMOS NAND2 gates. This is attributed to the smaller voltage swing of the CGaAs technology, 1.3 V for CGaAs versus 3.3 V for CMOS. The propagation delays of the CMOS gates are slightly faster than those of the CGaAs gates. This is somewhat surprising given the greater carrier mobilities of gallium arsenide. However, a closer inspection reveals that the mobility advantage is negated by the high device thresholds. This becomes obvious by looking at the basic saturation equation for drain current of a CMOS MOSFET transistor,

$$i_D = \frac{1}{2}\mu C_{OX}\left(\frac{W}{L}\right)(v_{GS} - V_T)^2 \qquad (1)$$

and that of a CGaAs MESFET transistor [62],

$$i_D = \beta(v_{GS} - V_T)^2 \qquad (2)$$

where,

$$\beta = \frac{2\varepsilon_s \mu v_{sat} W}{b(\mu V_{po} + 3v_{sat}L)} \qquad (3)$$

The transistors were sized such that *W/L* was the same for the CMOS and CGaAs circuits. Equations *1* and *2* are proportional to μ and $(v_{GS} - V_T)^2$. The higher supply voltage and lower thresholds of CMOS result in a $(v_{GS} - V_T)$ that compensates for the inferior carrier mobility of silicon. This is demonstrated quantitatively in Table 7.11. A 0.5 μm CGaAs n-type transistor is compared with a 0.35 μm CMOS n-type transistor. Both devices have a W/L ratio equal to one. The gates and drains are connected to appropriate supply voltages,

	0.5 µm CGaAs	0.35 µm CMOS
VDD (V)	1.3	3.3
V_T (V)	+0.65	+0.55
$I_{D,sat}$ (mA)	0.105	0.304
I_{GATE} (A)	5.655×10^{-8}	1.268×10^{-21}

Table 7.11: Characteristics of CGaAs and CMOS n-type transistors.

while the sources are grounded. The CMOS transistor has a drain current that is three times that of the CGaAs device. Another interesting phenomenon is the gate current. The gate current of the CMOS transistor is negligible. On the other hand, the CGaAs transistor exhibits measurable gate current, on the order of tens of nanoamps. For larger transistors the gate current may have a significant impact upon performance.

7.2.2.3 Standard Cell Library

A standard cell design approach was taken to increase the probability of a functional FXU II and speed the release of the prototype microprocessor. While a full-custom design methodology offers higher performance and reduced die size, the development time of a full-custom microprocessor would severely impact the schedule of the PUMA research project. The standard cell design methodology has several advantages. The physical design of the entire microprocessor depends on the layout of only a handful of logic gates, reducing the amount of manual effort. The reliability increases because each circuit can be fully characterized. The low overhead associated with implementing the standard cell library allows rapid development of complex ICs with only a few design engineers.

The PUMA CGaAs standard cell library[1] consists of thirty fully complementary logic gates; including most boolean functions of two and three variables, assorted complex functions, multiplexors, standard buffers, tristate buffers, and flip-flops. Most gates are available in drive strengths from 1x to 6x, while the inverters, standard buffers, and tristate buffers are available with drive strengths up to 64x. Table 7.12 lists some of the common gates available in the library along with the associated area, power, and delay values. The standard cell height was fixed at 48.05 μm, corresponding to the vertical dimension of a 6x inverter. The width varied depending upon the complexity of the logic gate. Care was taken with critical cells such as the D flip-flop to ensure that the width was kept to a minimum. Gate delays range from 137 picoseconds to 676 picoseconds, with a typical gate delay being approximately 350 picoseconds. Not included in this table are the vast assortment of and-or-invert and or-and-invert complex gates which have a greater delay. Also

gate	width (μm)	area (μm^2)	Power (μW)	t_{delay} (psec)
AND2	17.40	836	91	349
INV	9.10	437	12	137
MUX2	28.95	1391	152	511
NAND2	12.85	617	29	166
NOR2	13.70	658	98	235
OR2	18.30	879	110	443
XNOR2	24.90	1196	124	304
XOR2	24.00	1153	46	676

Table 7.12: PUMA 0.5 μm CGaAs standard cell characteristics.

1. The design of the PUMA CGaAs standard cell library was the work of several design engineers, most notably Phiroze N. Parakh and Keith L. Kraver.

omitted from the table are the D flip-flop elements which have a clock to Q delay of approximately 1 nanosecond.

The assortment of standard cells available in the library is specially suited to the CGaAs technology. Note the disparity between the NAND and NOR logic families. While the CMOS technology allowed the NAND and NOR logic styles to achieve similar levels of performance, the CGaAs technology favors the NAND logic style. The n-type transistors are faster than the p-type transistors by a large margin. To mitigate this effect only two p-type transistors may be connected in series, while up to four n-type devices may be connected in series. The library contains an assortment of NAND logic gates, including two-, three-, and four-input nand gates, but only two-input NOR and OR gates are available.

The fanout sensitivity for CGaAs complementary circuits is shown in Figure 7.13. The plot represents the results of HSPICE simulations of CGaAs standard cells driving similar gates. The intrinsic delays and fanout sensitivities are listed in Table 7.13. Comparing these results with those of the CMOS standard cell library from Section 7.1.2.3, we see again that the intrinsic delay of CGaAs logic gates is almost twice that of CMOS gates. The slopes indicate that the complementary CGaAs logic gates have higher fanout sensitivities than complementary CMOS logic gates. Again, the main factor limiting the performance of the CGaAs circuits is the high device thresholds. The increased fanout sensitivity can also be attributed to second-order effects, such as the increased gate capacitance and gate current associated with each additional load.

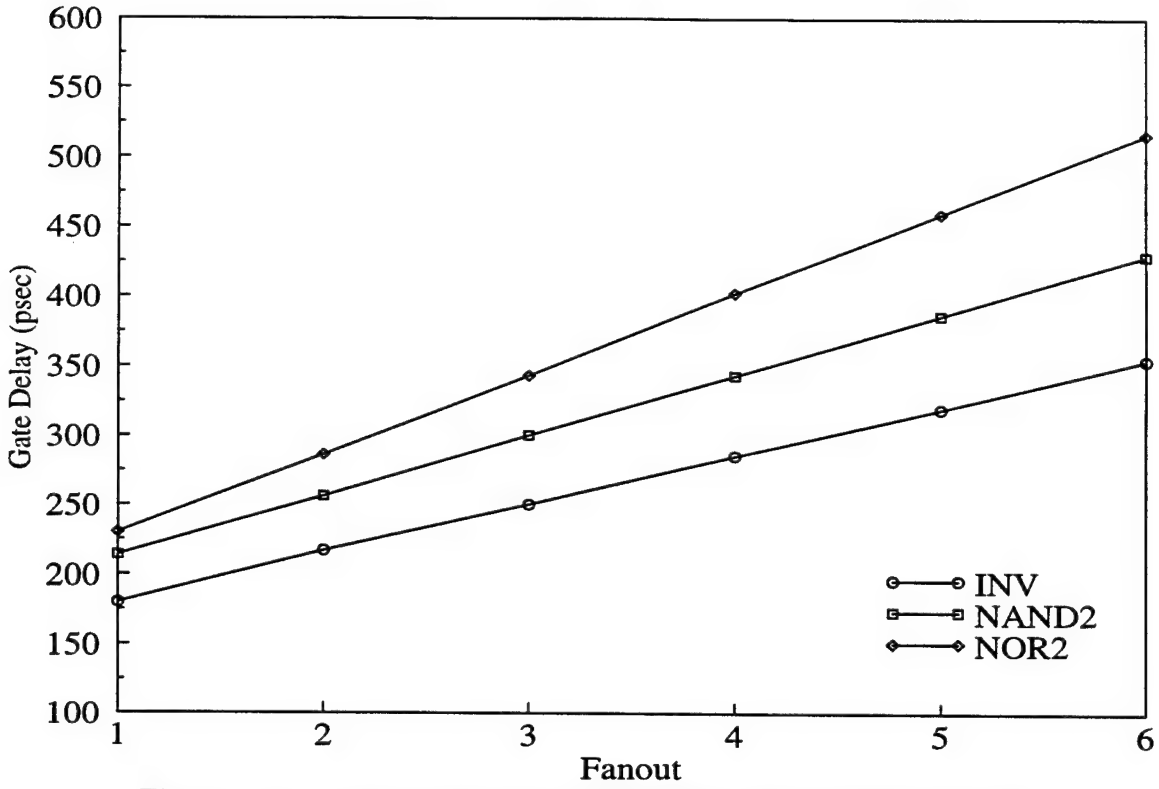


Figure 7.13: Fanout sensitivity of complementary CGaAs circuits.

	INV	NAND2	NOR2
Intrinsic delay (psec)	146	171	172
Fanout sensitivity (psec/load)	35	43	57

Table 7.13: Delay and fanout sensitivity of complementary CGaAs circuits.

Table 7.14 lists all of the logic gates in the CGaAs standard cell library and the corresponding usage in the FXU II design. The D flip-flop (DFF) and tristate buffer (TRIBUF) instance counts are grossly inflated because, as previously mentioned, the register file is based on D flip-flop cells and tristate buffers rather than a 6-T memory cell, as was the case in FXU I. A total of 1536 flip-flops and 3072 tristate buffers were used in the register file to construct the 48 x 32-bit array.

Gate Type	Instance Count	% of Total	Gate Type	Instance Count	% of Total
AND2	605	3.21	NAND2	331	1.76
AND3	254	1.35	NAND3	412	2.19
AND4	162	0.86	NAND4	140	0.74
AO21	97	0.51	NOR2	372	1.97
AOI21	27	0.14	OA21	31	0.16
AOI22	11	0.06	OA211	16	0.08
BUF	2337	12.39	OA22	51	0.27
DFF	4185	22.20	OA222	13	0.07
DFF_C	222	1.18	OAI21	89	0.47
DFF_CQ	3	0.02	OAI211	30	0.16
DFF_Q	16	0.08	OAI22	45	0.24
INV	808	4.29	OR2	392	2.08
MUX2	3055	16.20	TRIBUF	3268	17.33
MUX3	527	2.80	XNOR2	263	1.39
MUX4	606	3.21	XOR2	487	2.58
			Total	18855	100.00

Table 7.14: Standard cell gate usage in FXU II.

7.2.3 Module Design Summary

Table 7.15 lists the FXU II individual module statistics. The transistor counts are consistent with the estimates made in Table 7.7. The resulting area of FXU II is quite large, due in part to the coarse geometries of the CGaAs process and the inefficiency of a standard cell design approach.

The register file, the largest module in FXU II, requires twice as many transistors as the FXU I register file and consumes 20% of the FXU II die area. The execution cluster,

which consists of the three execution units, the ALU, BRU, and LSU, occupies 16% of the die. Since FXU II does not take advantage of the out-of-order superscalar characteristics of the proposed architecture, the execution units could have been integrated into a single unit. This would have significantly reduced the number of transistors and the area required. Such a design would also have increased the computational efficiency of the design, since in the FXU II execution unit as many as three-quarters of the transistors may be idle in a given cycle.

Module	Dimensions width x length (μm)	Area (mm^2)	Transistor Count	Power Dissipation (mW)
Instruction Fetch	2871 x 1952	5.607	23,816	113
Icache	4453 x 1688	7.521	66,076	376
Decoder	2062 x 2243	4.627	15,863	92
Scheduler	2492 x 885	2.208	10,886	40
Reorder Buffer	1296 x 859	1.115	7,248	39
Register File	5943 x 4954	29.451	120,298	634
Integer Unit	2258 x 2377	5.370	23,939	106
Branch Unit	2496 x 2564	6.402	26,508	144
Load Store Unit	4262 x 2601	11.089	49,725	261
Execution Cluster	9163 x 2688	24.638	100,172	511
Writeback	1690 x 1962	3.318	9,907	51
Bus Interface Unit	1083 x 3549	3.844	21,029	125
FXU II	13071 x 11416	149.229	383,027	2100

Table 7.15: FXU II Module Statistics.

7.2.4 FXU II Performance Summary

Static timing analysis played a key role in the development of FXU II. The EPOCH static timing analysis tool, TACTIC, utilized in FXU I was customized[1] to deal with the CGaAs standard cells. TACTIC computes delays by performing a delay table lookup based upon capacitive load and input slew rate. The delay tables are derived from HSPICE simulations of the individual CGaAs logic gates. The main concern was that the results of the TACTIC analysis be consistent with HSPICE simulations. Static timing analysis was performed on the entire FXU II design and the critical paths were simulated with HSPICE to verify consistency.

7.2.4.1 Global Path Summary

Figure 7.14 plots a distribution of the critical paths in FXU II in terms of the number of gate delays, or logic levels. While the number of logic levels along a given path does not correspond directly to actual delay, there is a close correlation. The clock to Q delay of a D flip-flop is included as a logic gate delay, since at approximately 1 nanosecond, it can be a large component of the total delay. The data is presented as a percent of the total number of paths. Approximately 70% of the paths in FXU II consist of fewer than nine gate delays. A small percentage of paths have more than 15 levels of logic. Only 0.2% of the paths have 20 gate delays, however, these paths could potentially limit the operating frequency.

---

1. Phiroze N. Parakh was responsible for customizing the TACTIC static timing analysis tool.

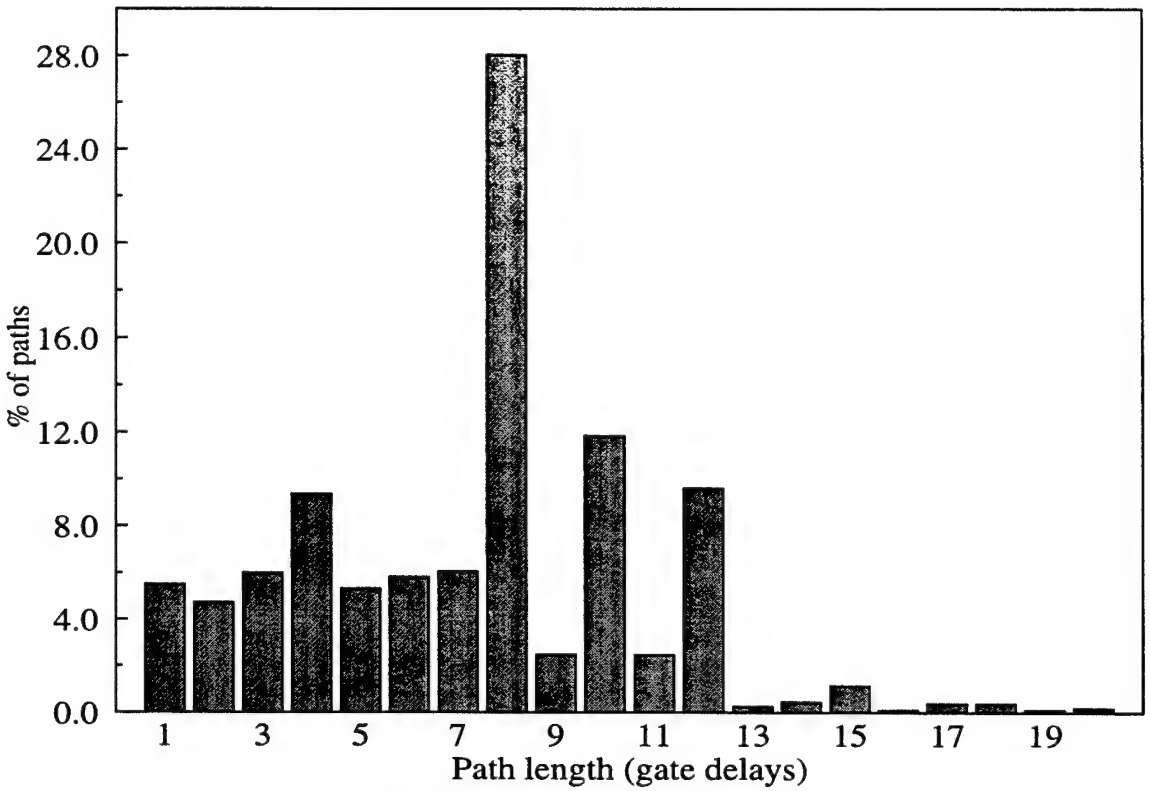


Figure 7.14: FXU II path distribution in terms of gate delays.

Figure 7.15 plots a distribution of the paths in FXU II based on total delay, as measured in nanoseconds. The data in Figure 7.14 is useful from a logic design perspective, but it does not take into account the proximity of gates on a logical path. A path may have relatively few gates, but span several modules and require a significant amount of time to propagate a value. Other paths that consist of many levels of logic may lie within a single module and have a relatively short delay. The distribution of Figure 7.15 shows that 60% of paths in FXU II have delays of approximately 5 nanoseconds or less, while 23% of the paths have delays between 7 and 9 nanoseconds. While less than 1% of the paths have a delay greater than 10 nanoseconds, these paths however, limit the operating frequency of FXU II to 87 MHz.

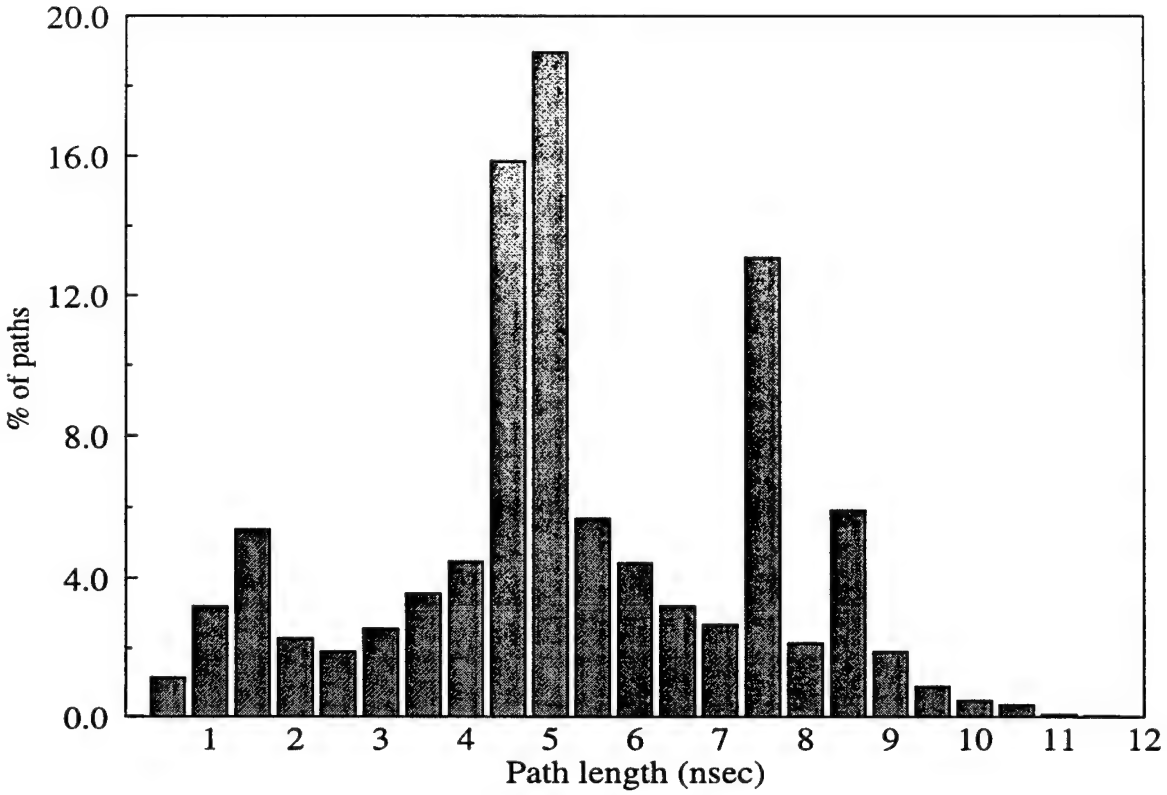


Figure 7.15: FXU II path distribution in terms of total delay.

A qualitative comparison of the distributions of paths in FXU II, as shown in Figures 7.14 and 7.15, against those of FXU I, as shown in Figures 7.6 and 7.7, indicates that FXU II is faster and simpler than FXU I. This indeed is the case. FXU II does not incorporate many of the high performance architectural features that are integrated into FXU I, such as instruction prefetching, branch prediction, and superscalar out-of-order execution.

7.2.4.2 Critical Path Analysis

In this section, the two most critical paths in FXU II will be analyzed in detail. A great deal more attention was paid to the critical paths in FXU II because it was the final version of the FXU architecture. The design environment for FXU II was customized from the

standard EPOCH framework so it was often necessary to verify that the results were being interpreted correctly and there was not an error in the database. The most critical path was extracted and simulated using HSPICE to verify that the static timing analyzer, TACTIC, was producing reasonable and consistent results.

The longest path in FXU II was found to lie in the branch resolution unit (BRU). The path consists of 20 levels of logic; TACTIC estimated the path delay to be 11.564 nanoseconds. Figure 7.16 shows a schematic view of the critical path and the corresponding HSPICE simulation output. The logic gates in the schematic have been assigned a gate number, preceded by the letter "g". The simulation waveform has been annotated to show the transition for the corresponding logic gate, the gate number is to the right of the rising or falling transition. The path begins with the rising edge of clock (CLK) at the reservation station D flip-flop, g0, and is buffered at the output of the reservation station by g1, and again at the input to the BRU logic by g2. Gates g3 through g5 decode the instruction to determine the select signal for mux g7, which in turn selects an input operand to form the branch target address. The target address is computed by a 32-bit carry-lookahead adder in gates g8 through g17. Finally, gate g18 buffers the output of the adder and g19 selects the input to the final D flip-flop, terminating the path.

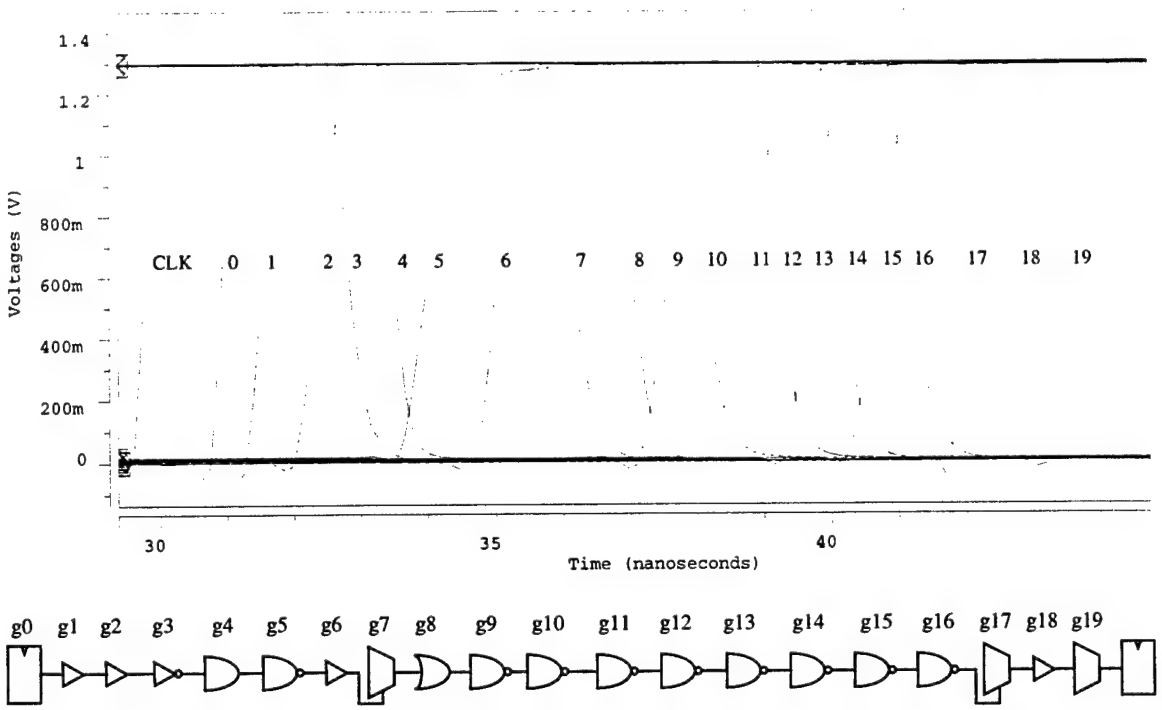


Figure 7.16: FXU II critical path: branch target address computation.

Table 7.16 presents a detailed timing analysis of the critical path. For each logic level, or gate instance, the table lists the drive strength of the gate and capacitive load present at the gate output. The table also compares the results of the HSPICE simulation with that of the static timing analysis performed by TACTIC. The absolute time, t, at which the output voltage reaches the midpoint, 1.65 V, and the delay of each individual gate, t_{gate}, is listed. Comparing the HSPICE simulation results to the static timing analysis revealed that the error in gate delay was found to be rather inconsistent, ranging from only 2% to a maximum of 95%, where a minimum-size gate drives a large load. However, HSPICE found the total path delay to be 13.710 nanoseconds, only a 19% increase over the static timing estimate. TACTIC estimated the maximum operating frequency of FXU II to be 86 MHz, but HSPICE simulations indicate it may be closer to 73 MHz.

Logic Level	0	1	2	3	4	5	6	7	8	9
Drive Strength	1x	3x	5x	1x	1x	1x	32x	2x	3x	1x
Load (fF)	48	323	821	101	78	66	3332	338	375	113
t (TACTIC)	0.947	1.490	2.312	2.676	3.170	3.448	4.177	5.203	5.975	6.387
t (HSPICE)	1.133	1.729	2.571	3.014	3.688	4.231	5.226	6.304	7.194	7.772
t_{gate} (TACTIC)	0.947	0.543	0.822	0.364	0.494	0.278	0.729	1.026	0.772	0.412
t_{gate} (HSPICE)	1.133	0.596	0.842	0.443	0.673	0.543	0.994	1.078	0.891	0.578

Logic Level	10	11	12	13	14	15	16	17	18	19
Drive Strength	2x	1x	2x	1x	2x	1x	2x	1x	6x	3x
Load (fF)	291	117	225	95	290	103	316	49	606	394
t (TACTIC)	6.767	7.229	7.604	7.920	8.250	8.619	8.975	9.614	10.332	11.564
t (HSPICE)	8.270	8.937	9.399	9.885	10.330	10.880	11.350	12.150	12.930	13.710
t_{gate} (TACTIC)	0.380	0.462	0.375	0.316	0.330	0.369	0.356	0.639	0.718	1.232
t_{gate} (HSPICE)	0.498	0.667	0.462	0.486	0.442	0.556	0.471	0.794	0.779	0.787

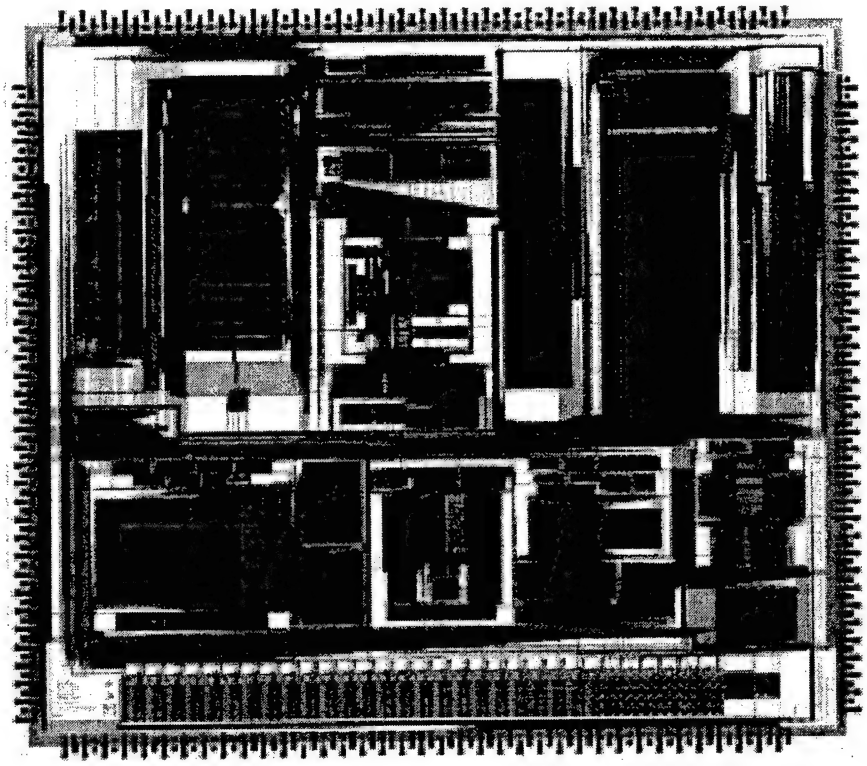
Table 7.16: Detailed timing analysis of FXU II critical path.

Several design modifications could have been made that would have reduced the critical nature of this path or eliminated it entirely. Gates g1 and g2 both buffer the same signal, creating in effect a redundant buffer. Eliminating g2 would save 800 picoseconds. Gates g3 through g5, which decode the instruction to determine the multiplexor select, could also be eliminated. This could be accomplished by hardwiring the multiplexor select to the decoded instruction word. The decoder, which translates PowerPC instructions to the internal format, could explicitly encode the multiplexor select in the internal instruction format. This design approach was taken with integer instructions in the ALU, and is the reason why a 32-bit add is not on the list of critical paths. Eliminating the decode gates from the path would save 1.6 nanoseconds. From an architectural perspective the entire path could be eliminated, or in essence relocated, by moving the target address computation to an earlier stage in the machine pipeline. Moving the computation may simply make

another path critical, but removing the redundant and inefficient gates as recommended would succeed in reducing the path to approximately 11.3 nanoseconds.

7.2.4.3 FXU II Design Statistics

The final physical layout and performance statistics for FXU II are presented in Figure 7.17. The die size is extremely large, 13.1 x 11.4 mm, owing in large part to the coarse CGaAs design rules and the inefficiency associated with constructing a microprocessor based entirely upon standard cells. The operating frequency is somewhat faster than FXU I, most likely due to the reduced complexity. The power is also lower than FXU I, due to fewer transistors and a lower supply voltage. FXU II is the first microprocessor developed in the Motorola's CGaAs process technology.



- 32-bit PowerPC
- 1 KB Icache / off-chip 16 KB Dcache
- 0.5µm CGaAs, 3 metal layers
- 383,027 transistors
- 13.1 x 11.4 mm die
- 86 MHz
- 2.1 W @ 1.3 V
- 288 pins (228 I/O)
- 391-pin PGA
- 250 area I/O pads
- Standard cell design
- Fully complementary circuit design

Figure 7.17: FXU II chip layout and design statistics.

7.3 Evaluation of CGaAs for Digital VLSI Applications

Based on the design of FXU II, an evaluation of the CGaAs process for digital VLSI applications can be made. The FXU II design example demonstrates that the digital CGaAs process could be improved in three areas of performance, namely speed, area, and level of integration. The operating frequency of FXU II is somewhat of a disappointment, given that gallium arsenide, like other III-V semiconductors, is noted for its high carrier mobility, and thus, higher operating frequencies. The current digital CGaAs process has negated the inherent speed advantage with high device threshold voltages. The FXU II die consumes a large amount of area. The 0.5 μm CGaAs process is actually a 0.7 μm process, enhanced with 0.5 μm channel lengths. The effect of scaling only the channel length results in a process that has relatively coarse design rules, as compared to the minimum feature size. The coarse geometries and low integration levels make it difficult to implement complex digital VLSI circuits in CGaAs.

The characteristics of CGaAs n-type transistors are compared to those of CMOS transistors in Table 7.11. The comparatively high device threshold was shown to significantly reduce the magnitude of the drain saturation current. Early process improvements were recommended that would lower the device thresholds to ±0.3 V. This enhancement would have a significant effect on performance, increasing (v_{GS} - V_T) in the saturation equation, resulting in a much higher saturation current. The effects of lower device thresholds are summarized in Table 7.17. Basically, the lower device thresholds affect device characteristics and circuit performance by approximately a factor of three. The saturation current increases by a factor of 2.5. Some circuit examples used in the chapter can be used to illus-

	Current 0.5 µm CGaAs	Enhanced 0.5 µm CGaAs
V_{Tn}/V_{Tp}(V)	+0.66/-0.53	+0.3/-0.3
$I_{Dsat,n}$ (µA/µm)	210	566
$f_{inv\text{-}ring}$ (MHz)	102.5	330.1
$P_{inv\text{-}ring}$ (mW)	0.511	1.434
$t_{inv\text{-}gate}$ (picoseconds)	157	49
FXU II frequency (MHz)	73	216
FXU II Power (W)	2.1	6.3

Table 7.17: Effects of lower device thresholds on CGaAs.

trate the potential increase in performance. HSPICE simulations demonstrated that the operating frequency of the ring oscillator would improve from 102.5 MHz to 330.1 MHz. As expected, the increased current also has an affect on power, essentially tripling the power dissipation from 0.5 mW to 1.4 mW. Finally, an HSPICE simulation of the critical path in FXU II estimates that the operating frequency would improve from 73 MHz to 216 MHz. The table also shows an estimate of the power dissipated by FXU II; the higher currents lead to three times the power consumption.

In lieu of higher performance, the lower device thresholds could enable a reduced die size. The transistor width could be decreased by a factor of 2.5, allowing the transistor to supply the same current as with the high thresholds and wider transistors. This would have a first-order effect on area and power, and a second-order effect on speed. Narrower transistors would allow the circuits to consume less area and the lower currents would result in reduced power consumption. The smaller area would decrease the interconnect length, reducing the RC delay and resulting in higher speed.

The area required for CGaAs digital VLSI circuits can be further reduced by process scaling. The industry has been aggressively scaling the CMOS process for many years now. The CGaAs process is falling far behind in terms of minimum feature size. There is no inherent limitation to scaling the CGaAs process, just the required capital investment. Table 7.18 compares the current 0.5 μm CGaAs process with a hypothetical scalable CMOS process described by Weste and Eshragian [63]. It is obvious that the dimensions of the current 0.5 μm CGaAs are not comparable to those developed for CMOS.

Gate metal and first-layer metal serve as local interconnect. The pitch of the CGaAs Schottky gate metal is twice that of CMOS polysilicon, while the first-layer metal pitch is 1.5 times larger. The CGaAs process does not require well regions to fabricate the complementary transistors, seemingly an advantage over CMOS. However, the advantage is negated by the large active area transistor spacing. These design rules, coupled with the increased transistor widths required because of the high thresholds, lead to very large cir-

Design Rule	Scalable CMOS	0.5 μm CGaAs
Gate pitch	4λ	8λ
Metal1 pitch	6λ	8λ
Metal2 pitch	7λ	8λ
Metal3 pitch	13λ	9.6λ
Contact	2λ	3.2λ
Via	2λ	4λ
Via2	2λ	4λ
Well spacing	6λ	na
Active area spacing	3λ	7.2λ

Table 7.18: Comparison of 0.5 μm CGaAs with scalable CMOS process.

cuits. The NAND2 and NOR2 standard cells in 0.35 μm CMOS and 0.5 μm CGaAs can be compared based on area as measured by λ^2. The comparison shows that the CGaAs standard cells consume approximately four times the area of the CMOS gates.

Physical inefficiencies also contribute to the increased area. An ohmic layer is required to establish a reliable electrical connection between the first-layer metal and the source and drain regions. This ohmic layer has been one of the key causes of poor yield. As a result, the design rules for ohmic are very conservative, requiring a large amount of area to establish a good connection. Another inefficiency has to do with the gate metal. In the CGaAs process, a conducting layer exists directly under the gate metal. If gate metal is used to directly connect the gates of n-type and p-type transistors, the isolation implant is blocked by gate metal, forming a conducting path between the channels of the two transistors. To prevent this, the gate metal must be strapped with first-layer metal to create an open circuit in the conducting path. The strap consists of two vias and a strip of metal, one strap is required for each pair of complementary transistors. These physical inefficiencies have a significant impact on the area of logic circuits.

Though high device thresholds, coarse local interconnect, conservative transistor spacing, and other physical inefficiencies result in large standard cells, cell interconnection and global interconnect do not appear to be a problem. Table 7.18 shows that the CGaAs design rules for second- and third-layer metal are comparable with those of CMOS. A visual inspection of the FXU II layout revealed that the design was not metal-limited, as was the case with the CMOS FXU I. This would likely change if the thresholds were

scaled, allowing the transistor sized to be reduced. However, the CGaAs could still benefit from reduced active spacing and finer local interconnect.

The proposed FXU architecture called for 500,000 transistors. This was deemed too aggressive for the CGaAs technology. As a result, the functionality had to be greatly simplified in order to meet the 400,000 transistor integer level. Building complex digital ICs, such as a superscalar microprocessor, is difficult with such a low level of transistor integration. While the integration level of CGaAs is the greatest perceivable challenge facing a microarchitect, the high thresholds and coarse geometries also limit the performance and cost of the circuit.

To be capable of supporting a complex digital IC, such as a single-chip superscalar microprocessor, CGaAs would have to support twice its current integration level, have its threshold voltages scaled, and have certain critical design rules reduced. Complex functions could be realized by distributing the design across several die. The PUMA project is an attempt to demonstrate the advantage of using MCM technology together with the CGaAs process, as a means of alleviating the penalty associated with partitioning the system across multiple chips.

7.4 Summary

The FXU architecture (FXU I) was implemented in a 0.35 μm CMOS process. FXU I, integrated 830 K transistors on a die which measures 9.9 x 9.9 mm. FXU I is expected to operate at 80 MHz and consume 4.1 W. FXU I implements a simplified version of the proposed architecture. Performance simulations estimate that the performance will be

0.62 IPC, a 3% improvement over a traditional pipeline. Initial test data indicate that the chip is functional.

FXU II was a 0.5 μm CGaAs implementation of the FXU architecture. The die measures 13.1 x 11.4 mm and integrates 383 K transistors. The chip is expected to operate at 86 MHz and consume 2.1 W from a 1.3 V supply. The constrained transistor budget and schedule issues resulted in fairly low performance. FXU II is only capable of supporting an execution rate of 0.51 IPC. However, this is the first microprocessor implemented in the CGaAs technology, and represents a significant advancement in the development of digital CGaAs circuits.

Currently it does not appear that CGaAs is capable of efficiently supporting a complex digital IC design, such as a single-chip superscalar microprocessor. The limited integration level does not allow the proposed 500,000 transistor superscalar architecture to be fully implemented. A simplified version of the architecture demonstrated other problems with CGaAs. The high device threshold voltages, coarse geometries, and physical inefficiencies resulted in a design that does not appear to be cost-effect or high-performance. Simply lowering the device thresholds to ±0.3 V could be translated into either a factor of three increase in operating frequency or reduction in transistor area. The CGaAs process would benefit from reduced thresholds, aggressive local interconnect, and increased integration levels.

CHAPTER 8

CONCLUSION

The goals of this research were to optimize the microarchitecture of a superscalar microprocessor with a limited number of transistors, to demonstrate the capabilities of the CGaAs technology through the design of a microprocessor, and to evaluate the CGaAs technology for digital VLSI circuits. Many advances were made to the state of digital CGaAs as a result of this research. The design of FXU II demonstrated the state of the CGaAs technology for implementing large digital integrated circuits. This chapter will summarize the contributions made by this research and outline areas for future work.

8.1 Research Contributions

This dissertation has made contributions in the areas of microarchitecture and circuit design. The most significant contribution is the optimization of a cost-efficient superscalar microarchitecture. This architecture was implemented in two PowerPC microprocessor implementations. CGaAs digital circuit techniques were explored and employed in the development of a high-speed PowerPC ALU test chip. The experience of designing several CGaAs VLSI circuits provided the basis to evaluate CGaAs for digital VLSI circuit applications.

8.1.1 Cost-efficient Superscalar Microarchitecture

While the academic community has done a great deal of research concerning high-performance microarchitectures, very little work has been done concerning small-scale implementations. Chapter 5 was an investigation into various high-performance features, and their applicability to small-scale design. The primary objective of this research was to optimize a cost-efficient microarchitecture for implementing a superscalar microprocessor using a transistor-limited semiconductor technology.

The proposed microarchitecture is characterized by a small on-chip cache, a small two-level dynamic branch predictor, an off-chip data cache, and a simple dual-issue superscalar execution scheme. The 1 KB on-chip instruction cache is backed by a two-entry stream buffer. The instruction fetch mechanism is guided by a 256-Byte two-level dynamic branch predictor implementing global sharing, capable of predicting branches with 87% accuracy. The 16 KB primary data cache is located off-chip, presenting a three-cycle memory access latency. The primary data cache latency is overcome by a pipelined load-store unit, load and store enhancements, and the superscalar nature of the execution core. The dual-issue out-of-order superscalar execution scheme requires only an eight-entry reorder buffer and two reservation stations per functional unit. The architecture makes use of a technique referred to as unit-operations to translate compound PowerPC instructions into fundamental RISC operations. The proposed architecture is expected to achieve an execution rate of 0.76 IPC, representing a 27% improvement over a comparable sequential pipelined microprocessor, and a 20% improvement over the baseline microprocessor configuration described in Section 5.1. The proposed architecture, running at 200 MHz, will achieve an estimated 153 MIPS.

8.1.2 PowerPC Microprocessor Implementations

The FXU architecture was implemented in a 0.35µm CMOS process. The chip, FXU I, integrated 830,000 transistors on a die which measures 9.9 x 9.9 mm. FXU I is expected to operate at 80 MHz and consume 4.1 W. FXU I implements a simplified version of the proposed architecture. Simulations estimate that the performance will be 0.62 IPC, a 3% improvement over a traditional pipeline. Initial test data indicate that the chip is functional.

The FXU architecture was also implemented in a 0.5µm CGaAs process. The die measures 13.1 x 11.4 mm and integrates 383,000 transistors. The chip is expected to operate at 86 MHz and consume 2.1 W from a 1.3 V supply. The constrained transistor budget, high device thresholds, and schedule issues resulted in fairly low performance. FXU II is only capable of supporting an execution rate of 0.51 IPC. However, this is the first microprocessor implemented in the CGaAs technology, and represents a significant advancement in the development of digital CGaAs circuits.

8.1.3 CGaAs Digital Circuit Design

Digital circuits were designed in Motorola's 0.5 µm CGaAs technology. The CGaAs technology was evaluated and contrasted with GaAs E/D MESFET technology. The critical issues relating to CGaAs circuit design were summarized. A study of digital CGaAs circuit techniques compared DCFL, complementary, and domino circuits against one another based on speed, power, area, and noise margin.

A PowerPC ALU test chip was developed based on CGaAs domino logic. The design consisted of 102,000 transistors and the die measured 5.8 x 5.3 mm. The design was opti-

mized for speed and power by using DCFL, complementary, and domino circuit techniques. Based on projected device models the chip is capable of operating at a frequency of 617 MHz, consuming 3.8 W with a 1.5 V supply. The design demonstrated both the potential of CGaAs domino circuits and the need for lower device thresholds to support high-speed circuit design. A 500,000 transistor implementation of the architecture was found to achieve a high degree of computation efficiency, 1.52 IPC/M-transistors, compared to contemporary microprocessor configurations.

8.1.4 Evaluation of CGaAs for digital VLSI Circuits

An evaluation of the digital CGaAs process for the implementation of digital VLSI circuits, such as microprocessors, was performed based on the design of FXU II. Several recommendations were made that would undoubtedly improve the process. Lower device thresholds would improve the speed of CGaAs circuits. Aggressive scaling of local interconnect and active area transistor spacing would reduce the area requirements of CGaAs logic circuits. Increased integration levels would allow complex circuits to be more readily implemented.

8.2 Future Work

While this thesis represents an investigation into CGaAs technology and limited-transistor superscalar microarchitectures, there are many possibilities for future research. The microarchitecture developed in this dissertation can extend beyond the CGaAs technology. Improvements could be made to the microarchitecture to further enhance performance. The ICs designed, FXU I and FXU II, can be fully tested and integrated into the proposed PUMA MCM system. Subsequent implementations of the FXU architecture can take

advantage of more aggressive circuit techniques and other optimization steps to improve the clock frequency.

8.2.1 Improving the Microarchitecture

This thesis developed a superscalar microarchitecture suitable for the CGaAs process technology. However, the microarchitecture is not inherently limited to CGaAs implementations. The concepts developed in this thesis can be applied to other technologies in which integration levels are limited, and to applications in which high-performance is desired, but cost considerations limit the resources available. The principles of superscalar execution and dynamic branch prediction have been shown to be effective at achieving high-performance with small transistor budgets.

The performance of the microarchitecture can be further enhanced by improving branch performance. The Branch Resolution Unit (BRU) should be integrated with the fetch mechanism, more closely resembling POWER and PowerPC implementations. This will allow branch predictions to be resolved sooner in the pipeline. The compiler hint bit was not utilized in the FXU architecture. The architecture could be enhanced by having the compiler set the hint bit to indicate branches that should be predicted using a static scheme. In this manner, the compiler could reduce interference in the branch predictor by identifying branches which can be accurately predicted using a static scheme. The FXU architecture can also be extended to dual-issue as the original proposed architecture specified.

Future implementations of the FXU may also improve performance by increasing the clock frequency. The clock frequency of FXU I and FXU II were limited by critical paths

that contained approximately 20 gate delays. Modern aggressive microprocessors have critical path lengths that are about half of this. The FXU architecture could be re-optimized for fewer gate delays. Aggressive circuit techniques, such as domino logic, could be used to decrease the cycle time in critical areas of the design. A good example of this is the PowerPC ALU test chip described in Section 2.2.6.

8.2.2 PUMA System

The two implementations of the FXU architecture can be fully tested and integrated into the design of the PUMA MCM system. At the time of this writing the CMOS FXU I has been packaged and is currently being tested. Initial tests indicate that the processor is functioning correctly. The CGaAs FXU II design was delivered to Motorola in July 1998, fabrication has been completed, and is currently being packaged. The CGaAs FXU II must also be tested. Concurrent with the packaging and testing of FXU II, the remainder of the PUMA system must be designed. The proposed system calls for a memory management unit (MMU), PCI controller, SRAMs, MCM substrate, printed circuit board (PCB), compiler, operating-system, and demonstration applications.

8.3 Epilogue

Many barriers are in place, preventing CGaAs from being a successful digital VLSI technology, such as feature size, integration level, wafer size, metallization, and yield. CGaAs lags years behind state-of-the-art commercial CMOS processes. These are not fundamental limitations; they could all be overcome with capital investment in equipment and process development. With the present technology, it is difficult to implement a complex digital VLSI circuit in CGaAs. This research has attempted to demonstrate the present

capabilities of the CGaAs technology through the design of the CGaAs PowerPC Microprocessor. This research involved optimizing a superscalar microarchitecture with a limited number of transistors. Circuit techniques for implementing digital logic in CGaAs were investigated. In the end, the processor did not achieve nearly the parallelism nor the speed of contemporary microprocessors. The redeeming features of CGaAs are radiation tolerance and low power. These properties may provide a niche market for radiation-hard low-power CGaAs digital VLSI circuits for space and satellite applications [11].

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J. M. Benedetto, "Economy-Class Ion-Defying ICs in Orbit," *IEEE Spectrum*, vol. 35, no. 3, pp. 36-41, March 1998.

[2] R. B. Brown, B. Bernhardt, M. LaMacchia, J. Abrokwah, P. N. Parakh, T. D. Basso, S. M. Gold, S. Stetson, C. R. Gauthier, D. Foster, B. Crawforth, T. McQuire, K. Sakallah, R. J. Lomax, and T. N. Mudge, "Overview of Complementary GaAs Technology for High-Speed VLSI Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 47-51, March 1998.

[3] R. B. Brown, P. Barker, A. Chandna, T. R. Huff, A. I. Kayssi, R. J. Lomax, T. N. Mudge, D. Nagle, K. A. Sakallah, P. J. Sherhart, R. Uhlig, and M. Upton, "GaAs RISC Processors," *14th Annual IEEE GaAs IC Symposium: Technical Digest 1992*, pp. 81-84, October 1992.

[4] M. Upton, T. Huff, P. J. Sherhart, P. Barker, R. McVay, T. J. Stanley, R. B. Brown, R. J. Lomax, T. N. Mudge, and K. Sakallah, "A 160,000 Transistor GaAs Microprocessor," *1993 International Solid-State Circuits Conference: Digest of Technical Papers*, pp. 92-94, February 1993.

[5] M. Upton, T. Huff, T. Mudge, and R. Brown, "Resource Allocation in a High Clock Rate Microprocessor," *ASPLOS VI Proceedings: Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 98-109, October 1994.

[6] K. Diefendorff and M. Allen, "Organization of the Motorola 88110 Superscalar RISC Microprocessor," *IEEE Micro*, pp. 40-63, April 1992.

[7] D. Dobberpuhl, R. Witek, R. Allmon, R. Anglin, D. Bertucci, S. Britton, L. Chao, R. Conrad, D. Dever, B. Gieseke, S. Hassoun, G. Hoeppner, K. Kuchler, M. Ladd, B. Leary, L. Madden, E. McLellan, D. Meyer, J. Montanaro, D. Priore, V. Rajagopalan, S. Samudrala, S. Santhanam, "A 200-MHz 64-b Dual-Issue CMOS Microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1555-1567, November 1992.

[8] M. LaMacchia, J. K. Abrokwah, B. Bernhardt, D. Foster, B. Crawforth, B. Mathes, T. McQuire, and T. Weatherford, "Radiation Hardened Complementary GaAs (CGaAS™)," *19th Annual IEEE GaAs IC Symposium: Technical Digest 1997*, pp. 59-61, October 1997.

[9] Meta-Software, Incorporated, *HSPICE User's Manual H9001*, Campbell, CA 95008, 1990.

[10] R. H. Krambeck, C. M. Lee, and H. Law, "High-Speed Compact Circuits with CMOS," *IEEE Journal of Solid-State Circuits*, vol. SC-17, no. 3, pp. 614-619, June 1982.

[11] T. D. Basso and R. B. Brown, "A Complementary GaAs Microprocessor for Space Applications," *The Second International Conference on Integrated Micro-Nanotechnology for Space Applications*, November 1998.

[12] K. Diefendorff, R. Oehler, and R. Hochsprung, "Evolution of the PowerPC Architecture," *IEEE Micro*, pp. 34-49, April 1994.

[13] S. Weiss, and J. E. Smith, *Power and PowerPC: Principles, Architecture, and Implementation*. Morgan Kaufmann, San Francisco, 1994.

[14] C. R. Moore, "The PowerPC™ 601 Microprocessor," *COMPCON Spring '93: Digest of Papers*, pp. 109-116, February 1993.

[15] C.R. Moore, D. M. Balser, J. S. Muhich, and R. E. East, "IBM Single Chip Processor (RSC)," *Proceedings of the 1992 International Conference on Computer Design*, pp. 200-204, October 1992.

[16] B. Burgess, M. Alexander, Y. Ho, S. Litch, S. Mallick, D. Ogden, S. Park, and J. Slaton, "The PowerPC™ 603 Microprocessor: A High Performance, Low Power, Superscalar RISC Microprocessor," *Spring COMPCON 94: Digest of Papers*, pp. 300-306, March 1994.

[17] K. Diefendorff and E. Silha, "The PowerPC User Instruction Set Architecture," *IEEE Micro*, pp. 30-41, October 1994.

[18] S. Song, M. Denman, and J. Chang, "The PowerPC 604 RISC Microprocessor," *IEEE Micro*, pp. 8-17, October 1994.

[19] D. Levitan, T. Thomas, and P. Tu, "The PowerPC™ 620 Microprocesor: A High Performance Superscalar RISC Microprocessor," *COMPCON '95: Digest of Papers*, pp. 285-291, March 1995.

[20] A. R. Kennedy, M. Alexander, E. Fiene, J. Lyon, B. Kuttanna, R. Patel, M. Pham, M. Putrino, C. Croxton, S. Litch, and B. Burgess, "A G3 PowerPC™ Superscalar Low-Power Microprocessor," *Proceedings of COMPCON '97*, pp. 315-324, February 1997.

[21] G. Gerosa, M. Alexander, J. Alvarez, C. Croxton, M. D'Addeo, A. R. Kennedy, C. Nicoletta, J. P. Nissen, R. Phillip, P. Reed, H. Sanchez, S. A. Taylor, and B. Burgess, "A 250-MHz 5-W PowerPC Microprocessor with On-Chip L2 Cache Controller," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 11, pp. 1635-1649, November 1997.

[22] L. Gwennap, "Comparing RISC Microprocessors," *Proceedings of the Microprocessor Forum*, October 1994.

[23] R. B. Brown, T. D. Basso, P. N. Parakh, S. M. Gold, C. R. Gauthier, R. J. Lomax, and T. N. Mudge, "Complementary GaAs Technology for a GHz Microprocessor," *18th Annual GaAs IC Symposium: Technical Digest 1996*, pp. 313-316, November 1996.

[24] B. T. Davis, C. Gauthier, P. Parakh, T. Basso, C. Lefurgy, R. Brown, and T. Mudge, "Impact of MCM's on High Performance Processors," *ASME Interpack 97*, pp. 863-868, June 1997.

[25] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*, 2nd ed., Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[26] Motorola, Inc., *PowerPC™ Microprocessor Family: The Programming Environments*, 1994.

[27] C. B. Hall and K. O'Brien, "Performance Characteristics of Architectural features of the IBM RISC System/6000," *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 303-309, April 1991.

[28] M. Johnson, *Superscalar Microprocessor Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[29] A. Cagney, PSIM: PowerPC Simulator, ftp://ftp.ci.com.au/pub/psim, 1994.

[30] M. K. Farrens and A. R. Pleszkun, "Improving Performance of Small On-Chip Instruction Caches," *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pp. 234-241, May 1989.

[31] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 364-373, May 1990.

[32] C. Melear, "The Design of the 88000 RISC Family," *IEEE Micro*, pp. 26-38, April 1989.

[33] N. P. Jouppi and D. W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 272-282, April 1989.

[34] M. D. Smith, M. Johnson, and M. A. Horowitz. "Limits on Multiple Instruction Issue," *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 290-302, April 1989.

[35] M. Butler, T. Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow, "Single Instruction Stream Parallelism is Greater than Two," *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pp. 276-286, May 1991.

[36] W. Hwu and Y. N. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pp. 18-26, June 1987.

[37] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, pp. 28-40, April 1996.

[38] J. E. Smith and A. R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," *Proceedings of the 12th Annual International Symposium on Computer Architecture*, pp. 36-44, June 1985.

[39] J. E. Smith and A. R. Pleszkun, "Implementing Precise Interrupts in Pipelined Processors," *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 562-573, May 1988.

[40] R. D. Acosta, J. Kjelstrup, and H. C. Torng, "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors," *IEEE Transactions on Computers*, vol. C-35, no. 9, pp. 815-828, September 1986.

[41] G. S. Sohi and S. Vajapeyam, "Instruction Issue Logic for High-Performance, Interruptable Pipelined Procesors," *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pp. 27-34, June 1987.

[42] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal*, vol. 11, pp. 25-33, January 1967.

[43] S. McFarling and J. Hennessy, "Reducing the Cost of Branches," *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pp. 396-403, June 1986.

[44] J. E. Smith, "A Study of Branch Prediction Strategies," *Proceedings of the 8th Annual International Symposium on Computer Architecture*, pp. 135-148, May 1981.

[45] T. Yeh and Y. N. Patt, "Two-Level Adaptive Training Branch Prediction," *The 24th ACM/IEEE International Symposium and Workshop on Microarchitecture*, pp. 51-61, November 1991.

[46] T. Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 124-134, May 1992.

[47] S. Pan, K. So, and J. T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 76-84, October 1992.

[48] T. Yeh and Y. N. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp. 257-266, May 1993.

[49] S. McFarling, "Combining Branch Predictors," *WRL Technical Note TN-36*, Digital Equipment Corporation, June 1993.

[50] J. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *IEEE Computer*, pp. 6-22, January 1984.

[51] C. H. Perleberg and A. J. Smith, "Branch Target Buffer Design and Optimization," *IEEE Transactions on Computers*, vol. 42, no. 4, pp. 396-412, April 1993.

[52] T. Yeh and Y. N. Patt, "A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution," *Proceedings of the 25th Annual International Symposium on Computer Microarchitecture*, pp. 129-139, December 1992.

[53] T. Yeh, D. T. Marr, and Y. N. Patt, "Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache," *Proceedings of the 7th ACM International Conference on Supercomputing*, pp. 67-76, July 1993.

[54] T. Yeh and Y. N. Patt, "Branch History Table Indexing to Prevent Pipeline Bubbles in Wide-Issue Superscalar Processors," *Proceedings of the 26th Annual International Symposium and Workshop on Microarchitecture*, pp. 164-175, December 1993.

[55] D. Lee, J. Baer, B. Calder, and D. Grunwald "Instruction Cache Fetch Policies for Speculative Execution," *Proceedings of the 22nd Annual International Sympsium on Computer Architecture*, pp. 357-367, June 1995.

[56] D. Kroft, "Lockup-Free Instruction Fetch/Prefetch Cache Organization," *Proceedings of the 8th Annual International Symposium on Computer Architecture*, pp. 81-87, May 1981.

[57] A. Klaiber and H. M. Levy, "An Architecture for Software-Controlled Data Prefetching," *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pp. 43-53, May 1991.

[58] T. Mowry, M. S. Lam, and A. Gupta, "Design and Evaluation of a Compiler Algorithm for Prefetching," *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operation Systems*, pp. 62-73, September 1992.

[59] T. Chen and J. Baer, "A Performance Study of Software and Hardware Data Prefetching Schemes," *Proceedings of the 21st Annual Internation Symposium on Computer Architecture*, pp. 223-232, May 1994.

[60] Cascade Design Automation, *EPOCH User's Manual, ver. 3.2*, Bellevue, WA 98006, 1995.

[61] A. R. Pleszkun and G. S. Sohi, "The Performance Potential of Multiple Functional Unit Processors," *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pp. 37-44, June 1988.

[62] S. I. Long and S. E. Butner, *Gallium Arsenide Digital Integrated Circuit Design*, McGraw-Hill Publishing Company, 1990.

[63] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective, Second Edition*, Addison-Wesley Publishing Company, 1993.